MICRO™

THE 6502 JOURNAL

No. 28 SEPTEMBER $2.00

MICRO™

September 1980
Issue Number 28

Staff

Editor/Publisher
Robert M. Tripp

Associate Editor
Mary Ann Curtis

Assistant Editor/ Advertising Manager
L. Catherine Bland

Circulation Manager
Carol A. Stark

Art/Advertising Coordinator
Terry Spillane

Software Consultant
Ford Cavallari

PET Specialist
Loren Wright

Comptroller
Donna M. Tripp

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824
Publication Number: COTR 395770

Subscription rates:U.S.	$15.00 per year
Foreign surface mail	$18.00 per year
Central America air	$27.00 per year
So. Amer/Europe air	$33.00 per year
Other air mail	$39.00 per year

For back issues, subscriptions, change of address or other information, write to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515

Articles

Departments

Southeastern Software 6414 Derbyshire Drive • New Orleans, LA 70126
504/246-8438 504/246-7937

Southeastern Software 'NEWSLETTER' for APPLE II Owners
NOW IN THE THIRD YEAR OF PUBLICATION
10 Issues per year for $10.00
Back Issues available at $1.00 each
EXAMPLE:
Send $10.00 and receive next 10 Issues
Send $30.00 and receive 30 Issues beginning with #2

DATA CAPTURE 3.0 - $29.95

Is DATA CAPTURE 3.0 just another Smart Terminal program? NO! It is a GENIUS Terminal program for use with the Micromodem II™. It will 'capture' ANYTHING that appears on the screen of your CRT. ANY program or data. If you are using the Source you can even 'capture' CHAT. There is no need to create files in your file space on the other system to transfer data to your Apple. If you can list it you can capture it.

* You can then SAVE the data to disk, dump it to your printer or even do simple editing with DATA CAPTURE 3.0.
* You can use DATA CAPTURE 3.0 to compose text off line for later transmission to another computer. Think of the timeshare charges this will save you!
* Use DATA CAPTURE 3.0 with the Dan Paymar Lower Case Adapter and you can enter UPPER or lower case from the keyboard for transmission to another system. You can also capture UPPER/lower case data from another system.
* A program is also included to convert your programs to text files for transmission using DATA CAPTURE 3.0.
* DATA CAPTURE 3.0 will save you money if you are using any timesharing system.

Requires DISK II™, Applesoft II™

Add $64.95 to order the Dan Paymar Lower Case Adapter

BAD BUY DISKETTE - $9.99

Of course it's a bad buy. If you have issues #2 thru #11 of the NEWSLETTER you can type these programs in yourself. Includes a couple of bonus programs.

Requires DISK II™, Applesoft II™

We ship within 3 working days of receipt of order and welcome your personal check. We also accept Visa and Master Charge.

LCMOD for PASCAL - $30.00

Finally! DIRECT entry of UPPER/lower case into the Pascal Editor. Why pay hundreds of dollars for a terminal just to set lower case entry with Pascal? If you have the Paymar Lower Case Adapter you can use this program.

* Left and right curly brackets for comment delimiters.
* An underline for VARs, program names and file names.
* The ESCape key does the shifting and Control O is used for ESCape. Have you ever typed in a page or two of text and lost it by hitting ESC accidentally? This won't happen with LCMOD.

Requires Language System and Paymar LCA

Add $64.95 to order the Dan Paymar Lower Case Adapter.

MAG FILES — $18.00

Finding it difficult to keep track of all those magazine articles you are reading? This program will help you do it. MAG FILES is Menu driven with separate modules for creating, editing, displaying and searching for your data. If you are using one drive a program is provided for transferring data to another diskette for backup. A sample data base of over 60 articles is included. The screen formatting and user orientation are what you have come to expect of Southeastern Software.

Requires DISK II™, Applesoft II™.

MAILER — $15.00

Don't let the low cost fool you. This is a single drive version of the program we use to maintain the NEWSLETTER subscriber list. Can be easily converted to 2.3 or 4 drives. Binary search and linear searches for finding any name in file. Sort on names and zip codes. Selective print by zip code or key. The separate modules are menu driven and will run on 32K system. There are 13 separate modules on the diskette for maintaining a mailing list. Sample data file included.

Requires DISK II™, Applesoft II™.

Games, Games, Games ....

"*Words, words, words! I'm so sick of words ....*" is the start of a song sung by Liza Doolittle in My Fair Lady. In this song she expresses her despair at the interest in words to the exclusion of other matters. I can fully understand her feelings. She feels that there is a lot more to life than just the "*words*" on which Professor Higgins spends all of his time. In a very similar vein, I feel that there is a lot more potential to the microcomputer than its popular use as an entertaining game-playing device. This is not to say that I am totally against computer games. Actually, I see nothing "illegal, immoral, or fattening" in using computer games for pure enjoyment. If a game can be used to interest people in the computer and/or help to teach them something, all the better. My complaint is that all too often, "The Game" is the exclusive use of the computer and the exclusive interest of the user.

I believe that the game glut poses two serious problems. First, I personally believe that one of the most fundamental problems of our modern society is the isolation of the individual. While there are obviously a large number of factors involved, the fact that individuals spend a large portion of their time watching television (the current figure just announced is 7.25 hours of television per day!) must be significant. It bothers me to see a majority of microcomputer users spending their time playing solitary computer games. While this is probably better than passively watching the TV, it does not do much to encourage social contact or interaction. Is the main impact of this fantastic microcomputer revolution going to be greater dependence on machine based interaction and less on interpersonal interaction? Given the natural interest in games, why not invent computer based games to be played by more than one individual. The computer could either be one more player (the elusive "fourth for bridge") or could provide a dynamic environment for games which are played exclusively by the human participants. While a few games are available along these general lines, by far the most common types of games are the one-on-one: one human against one microcomputer.

The second aspect of my "Games, games, games" complaint is that there are so many other uses of the microcomputer waiting to be discovered, but most of the potential discoverers are too busy playing games to consider alternative uses. Somewhere in the vast pool of new computerists there must be some individuals who could become the Einstein of the computer world. There is room for revolutionary improvements in the programming and application of computers. If the new computerists, who are being introduced to the microcomputer via games, get trapped into the game playing habit, then who will make the new discoveries and exciting improvements?

I have no simple solution. Since computer games are fun, many people are going to spend all of their computer time and money playing them. MICRO is going to be starting several series of articles in the coming months that will try to show how productive work can be as exciting and challenging as games, and vastly more rewarding. In the meantime, you should seriously consider how you are using your equipment, your time, and your money. Isn't it perhaps time that you started contributing to this field, instead of just playing around in it?

Robert M. Tripp

MICRO in the Lab
Cover Artist
Terry Spillane

Is that a crown that Lana is wearing? What did our Simian ancestor do to receive such royal treatment? Lana has demonstrated the rudiments of linguistic competence — the "crown" is an array of electronic sensors which are used to learn more about the phenomenon Lana has displayed.

While our Lana is fictitous, the scene depicted on our cover is based on an actual experiment. A chimpanzee named Lana has been taught to communicate with a small computer, using a special picture language.

Using a computer as the medium for the picture language, the designers of the Lana experiment have attained some significant advantages. The computer allows 24-hour monitoring and mass data storage. Only ten years ago, the equipment needed for this experiment would have taken up half a room and it would have cost over $10,000. Now, however, a simple single board Microprocessor (like the KIM-1) has more than enough processing power for such a task. Perhaps even more importantly, the computer can easily analyze sentences in a phrase-structure language for correct form. Actually, your Micro does this each time you run a basic program!

The Microcomputer's place in the lab has become well established; the Lana experiment is just one example. The next few years should see Micros being used in even more innovative ways in the lab...perhaps soon a Micro may even be generating, rather than monitoring, language!

μ

Creating Shape Tables, Improved!

Bulliding a Shapa Tabla for usa with tha Appla HI-RES Graphics can ba a painful task. This Improvad Shape Making Routine turns the pain into plaasure.

Peter A. Cook
1443 N. 24th St.
Mesa, AZ 85203

Three cheers to John Figueras for unraveling the mysteries of the Apple shape table in MICRO 19:11. His article presented an extremely useful tool for creating shapes, and greatly simplified a task which had been so difficult and time comsuming as to be hardly worth the effort.

After using the Figueras programs a few times, it became apparent that they would be much more convenient if they were combined into one large program. Also, they contained several minor errors which needed correcting. This article describes changes and corrections which greatly increase the usefulness of the original programs.

Combined Program

Combining the programs for initializing, creating, and displaying shape tables into one large program eliminates the need for typing the name of the next program each time you need to load it from the disk, and then waiting for it to be loaded. It also eliminates the need to continually re-enter the name and starting address of the desired shape table, and the subsequent wait for it to be loaded.

I combined the three original programs by treating them as subprograms. Since they all used similar line numbers, they required extensive renumbering. This was easy to accomplish using the 'Renumber' program found on the DOS 3.2 master diskette. The numbers were not done consecutively in order that the last two digits would remain the same as in the original programs in most cases. The new line numbers correspond to the old ones roughly as follows: title page, 100-150; initialize, 1000-1300; create shapes, 2000-3300; and display shapes, 4000-4500.

Title Page

The program begins by listing the title information and then by automatically loading the numerals shapefile. I have used the term 'shapefile' throughout the program to denote a shape table which has been stored as a disk file, as opposed to one which merely resides in RAM.

The program has been converted for use with a single disk drive by omitting the volume and drive numbers from the disk commands, because with the prices of drives being what they are, I would venture to say that most of us have only one.

A short menu then appears, which allows selection of any of the three subprograms, or termination of the program.

Initializing Subprogram

The greatest change made in this subprogram was the removal of the steps for producing the cursor. Placing the cursor into every shape table as the first shape in each one was wasteful of space, and very confucing . The cursor is always available as the first shape in the numerals shapefile, which is loaded when the program begins. Details of the numerals shapefile will be covered later. By using the improved program, your tables will now contain only the desired shapes, and will start with number one instead of number two.

Because of the removal of the cursor, line 1060 now adds a few more zeros so that the starting address of the first empty shape will contain a zero end-of-record mark. Line 1090 now calculates the index to the first shape instead of to the cursor. The variable ADDR had to be changed to ASVE to make it compatible with the shape creating subprogram.

Lines 1260-1300 were changed to let you know that the computer is doing what it is supposed to do, and to ask if you want to save the file on disk at this time. You can save time by waiting until the end of the shape creating subprogram before storing the shape table on disk.

The menu is then repeated at the

bottom to avoid having to return to the title page.

Shape Creating Subprogram

This subprogram assumes that you are still working with the same shape table that you initialized in the previous subprogram, and shows you what its name and starting address are. In case you want to work on a different shapefile which was previously stored on the disk, allowance is made for entering its name and address. The desired shape table is then loaded into its proper location.

The computer then checks to see if there is any space left for more shapes in that table. If not, it so advises you and tells you the address of the next free byte after the end of the table. The original program attempted to do this, but actually it accessed the first two bytes of the cursor vectors instead of finding the zero end-of-record mark, and thus provided a meaningless number. Lines 2132-2262 include the changes to correct this.

Since the cursor is now located in a different shape table than the one with which you are currently working, the computer must be able to switch from one table to the other as needed, to line 2264 remembers the pointer for the new shape table, and uses it again in the line 3170.

The text at the bottom of the plotting grid has been improved by adding line 2350 to show the number fo the shape you are currently working on. The limits of the starting coordinates are shown in lines 2360-2380, along with the fact that coordinates are measured from the upper left. Error checks were added to prevent entering coordinates located outside of the grid, which could stop the program in some instances.

The word "ERASE" was added to the list of keyboard commands LEFT, RIGHT, etc. In the original program, no checks were made on the values of x and y when entering L, R, U, or D, so if you accidentally exceeded certain grid boundaries the program would shut down. This was especially easy to do if you were using the "repeat" key to move the cursor. Lines 2600-2664 now contain error checks which prevent the grid limits from being exceeded, and sound a beep if you attempt to do so.

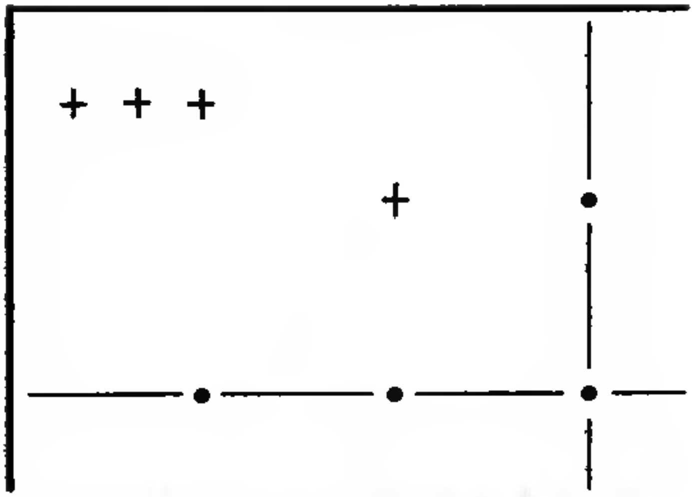


Figure 1. **Upper left corner of the display grid, showing the starting points for the three possible digits of the shape number, and for the shape itself.**

The original program placed a permanent cursor mark in the starting position. This meant that there were always two cursor marks visible within the grid, which was sometimes confusing. Line 2390 now places a large " + " in the starting square, the points of which are always visible around the outside of the cursor or around the outside of a plotted circle. The original program also attempted to give a reverse image of the cursor if it passed through a plotted circle. The succession of XDRAW commands,

```
]
*4E20.4EDB

4E20- 0B 00 18 00 1E 00 30 00
4E28- 3D 00 4C 00 5D 00 6D 00
4E30- 7D 00 8D 00 9B 00 AB 00
4E38- 3E 24 2D 36 04 00 DB DB
4E40- DB 24 0C 2D 15 17 35 36
4E48- 1E 3F 0F 18 0D 18 27 00
4E50- DB DB DB 08 5B 0D 18 36
4E58- 36 F6 2D 04 00 DB DB DB
4E60- 08 18 0C 2D 15 F6 BF 17
4E68- 2E 2D 25 00 DB DB DB 08
4E70- 18 2B 2D 35 1E 1E AD F6
4E78- 3F 0F 18 04 00 DB DB DB
4E80- 2E 2D B5 23 0C 18 24 BC
4E88- 0A 18 17 04 00 DB DB DB
4E90- 12 0E 2D 0D 18 24 1C 3F
4E98- 27 2C 2D 25 00 DB DB DB
4EA0- 32 0E 2D 0D 18 E4 3F 27
4EA8- 0C 0C 2D 04 00 DB DB DB
4EB0- 08 18 2B 2D 35 1E 1E 1E
4EB8- 36 04 00 DB DB DB 20 0C
4EC0- 2D 15 F6 3F 17 76 2D 0D
4EC8- 1B 24 00 DB DB DB 92 2D
4ED0- 0D 18 0D 1B 24 E4 3F 17
4ED8- 76 2D 04 00
```

Figure 2. **Hex pairs of the numerals shape table.**

however, was incorrect for all combinations of plotting, erasing, and passing through the starting position. Changes were made in lines 2680, 2740, 3040 to correct this. Now it is always obvious where the cursor is located and where the starting position is located.

The erase command is only effective immediately following a plot command. There is a way to erase any other plotted point, however, and that is by simply plotting over top of a point which has already been plotted. This will not erase the circle plotted in the grid, but the point will not appear in the finished shape when it is drawn to the right of the grid after the quit command.

In the original program, the warning "SHAPE TABLE FULL WITH THIS SHAPE" appeared both after the second-last shape as well as after the last shape. Changing N to N-1 in line 3230 allows the warning to appear only after the last shape.

The menu is repeated again at the bottom to allow selecton of any other subprogram, to to run the same one again.

Shapa Display Subprogram

This subprogram starts out as the previous one did, by listing the name and address of the shapefile you are currently working with. If you wish to display a different one, enter its name and address.

Some variable names were changed to keep them compatible with the rest of the program. ADDR was changed to ASVE, and NN was changed to N. In line 4114 (line 70 in the original) NL was changed to NLO, although either variable is acceptable since Applesoft only recognizes the first two characters of a variable name.

In the original program the screen went black after the shapefile was loaded, and you had to remember to press any key to start the display. Line 4150 keeps the instruction on the screen until you need it, and line 4202 takes you immediately into the first page of the display.

The grid lines created by the original program had an odd dot pattern which was not very useful because it didn't show where the starting positions of the shapes were located. Lines 4250-4310 were changed to present the dot pattern shown in Figure 1.

Pressing any key after the last page of the display puts the menu back on the screen.

Numarals Shapefile

In order to use the above program, the 'numerals' shape table must already have been stored on the disk in order to have the cursor available. If this has not been done, it will be necessary to load the shape table using either of the two following methods.

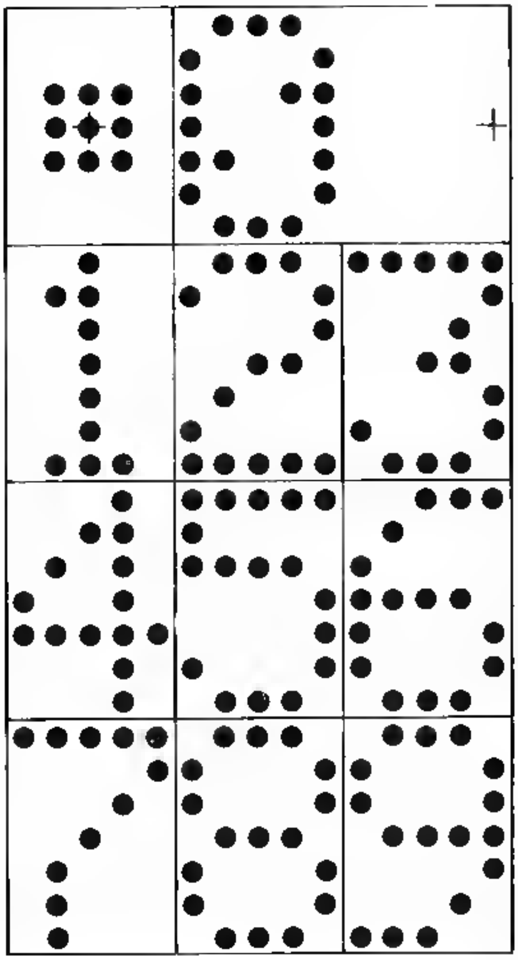


Figure 3. Cursor and numerals. The **starting point is in the center of the cursor, and fiva spaces to tha right of all the numerals.**

Figure 2 lists the hex values of the entire numerals shape table. It can be placed in RAM by entering the monitor mode, typing the addresses at the left, such as 4E20, followed by a colon, followed by each two-character element separated by a space. Since there are 188 elements, this may take some time. When you have finished, don't forget to save what you just typed before you run the program. Use BSAVE SHAPEFILE NUMERALS, A20000, L188.

The numerals are of the same design as the Apple numerals and are depicted in Figure 3. The starting point was placed five spaces to the right of each numeral, so that the finished numeral will be shifted off to the left of the shape which is displayed in the same block with it.

Another method is to type in just enough of the shape table to have the cursor available, and then to form your own numerals by using the shape creating subprogram. To do this, POKE each of the values in Figure 4 into its proper location by using the format POKE 20000, 1. Transfer it to the disk using BSAVE SHAPEFILE NUMERALS, A20000, L30. Then run the program and select the shape creating subprogram. Enter the name SHAPEFILE NUMERALS, and the address 20000. Form all of the digits in the order zero through nine by following the instructions on the screen, and then you will be ready to create and display other shape tables.

Conclusion

The program listing is presented on the following pages. In order to save space, all remarks were removed except for a title at the beginning of each subprogram. Basically, the same remarks apply as published in the original article.

In closing, I would like to thank John Figueras for providing Apple users with a most useful addition to their repertoire of utility programs.

μ

Location	Value	Description
20000	1	Number of shapes completed
20001	0	
20002	24	Location in table, starting address + 24
20003	0	
20024	62	Cursor vectors
20025	36	Cursor vectors
20026	45	Cursor vectors
20027	54	Cursor vectors
20028	4	Cursor vectors
20029	0	Zero end-of-record mark

Figure 4. **Minimum entries for producing the cursor in the numerals shape table.**

Major Peter Cook is a jet pilot instructor at Williams Air Force Base in Arizona. He uses his Apple II to simulate aircraft scheduling problems at work, and designs games for his kids at home. This is his second article for MICRO.

```
]LIST

100  REM SHAPEFILE CREATE/DISPLAY
          P. COOK, JAN 1980
        ADAPTED FROM J. FIGUERAS
        MICRO MAGAZINE, DEC 1979

110  HOME : PRINT : PRINT "******
     ****************************
     ******"
112  HTAB 9: PRINT "SHAPEFILE CRE
     ATE/DISPLAY"
114  PRINT : HTAB 12: PRINT "P. C
     OOK, JAN 1980"
116  HTAB 9: PRINT "ADAPTED FROM
     J. FIGUERAS"
118  HTAB 9: PRINT "MICRO MAGAZIN
     E, DEC 1979"
120  PRINT : PRINT "*************
     ***************************"

122 D$ =  CHR$ (4): PRINT D$;"NOM
     ON C,I,O": PRINT D$;"BLOAD S
     HAPEFILE NUMERALS"
130  VTAB 13: HTAB 6: PRINT "1  I
     NITIALIZE SHAPEFILE"
132  PRINT : HTAB 6: PRINT "2  CR
     EATE SHAPES"
134  PRINT : HTAB 6: PRINT "3  DI
     SPLAY SHAPES"
136  PRINT : HTAB 6: PRINT "4  EN
     D"
140  VTAB 23: INPUT "SELECT (1/2/
     3/4)? ";IN$
144  IF  VAL (IN$) < 1 OR  VAL (I
     N$) > 4 THEN  VTAB 23: HTAB
     19: PRINT "      ": GOTO 140
146  ON  VAL (IN$) GOTO 1010,2010
     ,4010,150
150  TEXT : HOME : END
1000  REM  INITIALIZE.
1010  TEXT : HOME : PRINT "INITIA
     LIZE NEW SHAPEFILE"
1020  PRINT : PRINT "  NAME OF NE
     W SHAPEFILE": INPUT "  ?";NA
     ME$
1030  PRINT : PRINT "  STARTING A
     DDRESS (DECIMAL)": INPUT "
     ?";ASVE
1040  PRINT : PRINT "  NUMBER OF
     SHAPES TO BE STORED IN FILE"
     : INPUT "  ?";N
1060  FOR I = 0 TO 2 * N + 3
1070  POKE ASVE + I,0: NEXT
1090 N = 2 * N + 2
1110  POKE ASVE + 2,N - 256 *  INT
     (N / 256)
1120  POKE ASVE + 3, INT (N / 256
     )
1260  PRINT : PRINT "SHAPEFILE IN
     ITIALIZED"
1280  INPUT "  SAVE ON DISK (Y/N)
     ? ";IN$: IF IN$ <  > "Y" THEN
     1310
1290  PRINT D$;"BSAVE ";NAME$;",
     A";ASVE;", L";N + 1
1300  PRINT : PRINT "SAVED"
1310  VTAB 21: PRINT "1 INIT   2
     CREATE   3 DISPLAY   4 END"
1320  GOTO 140
2000  REM  CREATE SHAPES.
2010 I = 0: TEXT : HOME : PRINT "
     CREATE NEW SHAPES IN SHAPEFI
     LE"
2020  PRINT : PRINT "CURRENT SHAP
     EFILE AND ADDRESS:"
2030  PRINT : HTAB 3: PRINT NAME$

2040  PRINT : HTAB 3: PRINT ASVE
2050  PRINT : PRINT : PRINT "FOR
     NO CHANGE, PRESS RETURN:"
2070  PRINT : INPUT "  DIFFERENT
     FILE? ";IN$: IF  LEN (IN$) =
     0 THEN 2080
```

```
2075 NAME$ = IN$:I = 1
2080  PRINT : INPUT "  DIFFERENT
     ADDRESS? ";IN$: IF  LEN (IN$
     ) = 0 THEN 2100
2085 ASVE =  VAL (IN$):I = 1
2100  IF I = 0 THEN 2130
2110  PRINT O$;"BLOAD ";NAME$;" ,
     A";ASVE
2130 MAX =  PEEK (ASVE + 2) + 256
      *  PEEK (ASVE + 3)
2132 FB = ASVE +  PEEK (ASVE + MA
     X - 2) + 256 *  PEEK (ASVE +
     MAX - 1)
2140 MAX = (MAX - 2) / 2
2160 N =  PEEK (ASVE)
2220  IF MAX > N THEN 2260
2222  IF  PEEK (FB) <  > 0 THEN F
     B = FB + 1: GOTO 2222
2224 FB = FB + 1
2230  PRINT : PRINT : PRINT "SHAP
     E TABLE FULL, NEXT FREE BYTE
      ";FB
2240  GOTO 1310
2260 INDEX =  PEEK (ASVE + 2 * N +
     2) + 256 *  PEEK (ASVE + 2 *
     N + 3)
2262 ADDR = ASVE + INDEX
2264 AHI =  INT (ASVE / 256):ALO =
     ASVE - 256 * AHI: POKE 232,A
     LO: POKE 233,AHI
2280 N = N + 1: POKE ASVE,N
2300  HCOLOR= 3: SCALE= 1: ROT= 0
     :CYCLE = 0
2310  HGR
2320  FOR X = 0 TO 150 STEP 10: HPLOT
     X,0 TO X,150: NEXT
2330  FOR Y = 0 TO 150 STEP 10: HPLOT
     0,Y TO 150,Y: NEXT
2350  HOME : VTAB 21: PRINT "SHAP
     E NUMBER ";N;" OF ";MAX
2360  PRINT "ENTER STARTING COORD
     S (UPPER LEFT 1,1)"
2370  INPUT "X (1-15)? ";X: IF X <
     1 OR X > 15 THEN 2370
2372 X = 10 * X - 5
2380  INPUT "Y (1-15)? ";Y: IF Y <
     1 OR Y > 15 THEN 2380
2382 Y = 10 * Y - 5
2390  HPLOT X,Y - 4 TO X,Y + 4: HPLOT
     X - 4,Y TO X + 4,Y:XS = X:YS
      = Y
2410  PRINT : PRINT : PRINT : PRINT

2420  PRINT "MOVE CURSOR WITH KEY
     S"
2430  PRINT "  L-LEFT  R-RIGHT  U
     -UP  D-DOWN"
2440  PRINT "  P-PLOT  E-ERASE  Q
     -QUIT"
2450  POKE 232,32: POKE 233,78
2460 KEY$ = "":KSVE$ = "": GOTO 2
     570
2480  IF FLAG = 1 THEN 2520
2500  XDRAW 1 AT X1,Y1
2520 X1 = X:Y1 = Y:FLAG = 0
2530  XDRAW 1 AT X,Y
2550 KI$ = KSVE$:KSVE$ = KEY$
2570  GET KEY$
2590  IF KEY$ <  > "U" THEN 2610
2600 SYMBOL = 0:Y = Y - 10: IF Y <
     5 THEN Y = Y + 10: GOTO 2664

2602  GOTO 2760
2610  IF KEY$ <  > "R" THEN 2630
2620 SYMBOL = 1:X = X + 10: IF X >
     145 THEN X = X - 10: GOTO 26
     64
2622  GOTO 2760
2630  IF KEY$ <  > "D" THEN 2650
2640 SYMBOL = 2:Y = Y + 10: IF Y >
     145 THEN Y = Y - 10: GOTO 26
     64
2642  GOTO 2760
2650  IF KEY$ <  > "L" THEN 2670
2660 SYMBOL = 3:X = X - 10: IF X <
     5 THEN X = X + 10: GOTO 2664


2662  GOTO 2760
2664  VTAB  PEEK (37): PRINT  CHR$
     (7): GOTO 2570
2670  IF KEY$ <  > "P" THEN 2690
2680 FLAG = 1: GOSUB 3000: GOTO 2
     520
2690  IF KEY$ = "Q" THEN 3100
2710  IF KEY$ <  > "E" THEN 2570
2720  HCOLOR= 0:FLAG = 0: GOSUB 3
     000
2740 KSVE$ = KI$: HCOLOR= 3: GOTO
     2530
2760  IF KSVE$ = "P" THEN SYMBOL =
     SYMBOL + 4
2780 CYCLE = CYCLE + 1
2790  IF CYCLE <  > 1 THEN 2810
2800 BYTE = SYMBOL: GOTO 2480
2810  IF CYCLE <  > 2 THEN 2900
2820 BYTE = BYTE + 8 * SYMBOL
2840  IF BYTE > 7 THEN 2480
2860 BYTE = BYTE + 8: POKE ADDR,B
     YTE:ADDR = ADDR + 1
2880 BYTE = 24:CYCLE = 2: GOTO 24
     80
2900  IF SYMBOL > 3 THEN 2930
2910 BYTE = BYTE + 64 * SYMBOL
2930  POKE ADDR,BYTE:ADDR = ADDR +
      1
```

```
2950  IF SYMBOL = 0 OR SYMBOL > 3
      THEN 2980
2970 CYCLE = 0: GOTO 2480
2980 CYCLE = 1:BYTE = SYMBOL: GOTO
     2480
3000  FOR Y2 = Y - 3 TO Y + 3 STEP
     6: HPLOT X - 1,Y2 TO X + 1,Y
     2: NEXT
3010  FOR Y2 = Y - 2 TO Y + 2 STEP
     4: HPLOT X - 2,Y2 TO X + 2,Y
     2: NEXT
3020  FOR Y2 = Y - 1 TO Y + 1: HPLOT
     X - 3,Y2 TO X + 3,Y2: NEXT
3040  RETURN
3080  IF KSVE$ < > "P" THEN 3150

3100  IF CYCLE < > 2 THEN 3120
3110  POKE ADDR,BYTE:ADDR = ADDR +
      1
3120  IF CYCLE < > 1 THEN 3140
3130 BYTE = BYTE + 32: GOTO 3150
3140 BYTE = 4
3150  POKE ADDR,BYTE:ADDR = ADDR +
      1
3170  POKE ADDR,0:ADDR = ADDR + 1
     : POKE 232,ALO: POKE 233,AHI
     : XDRAW N AT 200,75
3180  INPUT "SAVE SHAPE (Y/N)? ";
     KI$
3190  IF KI$ = "Y" THEN 3220
3200 N = N - 1: GOTO 2260
3220 N = N + 1:ADDR = ADDR - ASVE

3230  IF N - 1 < MAX THEN 3270
3240  PRINT "WARNING - TABLE FULL
      WITH THIS SHAPE"
3250  IF N > MAX THEN 3310
3270  POKE ASVE + 2 * N,ADDR - 25
     6 *  INT (ADDR / 256)
3280  POKE ASVE + 2 * N + 1, INT
     (ADDR / 256)
3290  INPUT "DONE (Y/N)? ";KI$
3300  IF KI$ = "N" THEN 2160
3310  INPUT "SAVE SHAPEFILE (Y/N)
     ? ";KI$
3330  IF KI$ = "Y" THEN 3360
3340  IF KI$ = "N" THEN 3370
3350  GOTO 3310
3360  PRINT D$;"BSAVE ";NAME$;",
     A";ASVE;", L";ADDR
3370  HOME : GOTO 1310
4000  REM  DISPLAY SHAPES.
4010 I = 0: TEXT : HOME : PRINT "
     DISPLAY SHAPES IN SHAPEFILE"

4020  PRINT : PRINT "CURRENT SHAP
     EFILE AND ADDRESS:"
4030  PRINT : HTAB 3: PRINT NAME$

4040  PRINT : HTAB 3: PRINT ASVE
4050  PRINT : PRINT : PRINT "FOR
     NO CHANGE, PRESS RETURN:"
4070  PRINT : INPUT "  DIFFERENT
     FILE? ";IN$: IF  LEN (IN$) =
     0 THEN 4080
4075 NAME$ = IN$:I = 1
4080  PRINT : INPUT "  DIFFERENT
     ADDRESS? ";IN$: IF  LEN (IN$
     ) = 0 THEN 4100
4085 ASVE =  VAL (IN$):I = 1
4100  IF I = 0 THEN 4114
4110  PRINT D$;"BLOAD ";NAME$;",
     A";ASVE
4114 NHI = 78:NLO = 32
4120 AHI =  INT (ASVE / 256):ALO =
     ASVE - 256 * AHI
4140 N =  PEEK (ASVE)
4150  VTAB 23: PRINT "PRESS SPACE
      BAR FOR EACH PAGE OF TABLE"
     : GET KEY$
4160  HGR : POKE  - 16302,0
4170  HCOLOR= 3: SCALE= 1: ROT= 0

4180  FOR I = 1 TO N
4190 IMOD = I - 36 *  INT (I / 36
     )
4200  IF IMOD < > 1 THEN 4350
4202  IF I = 1 THEN 4240
4210  GET KEY$
4240  CALL 62450
4250  HPLOT 0,0 TO 270,0 TO 270,1
     80 TO 0,180 TO 0,0
4260  FOR L = 45 TO 225 STEP 45
4270  FOR J = 15 TO 165 STEP 15
4280  HPLOT L,J
4290  NEXT J,L
4300  FOR L = 30 TO 150 STEP 30
4310  FOR J = 15 TO 255 STEP 15
4320  HPLOT J,L
4330  NEXT J,L
4350  IF IMOD = 0 THEN IMOD = 36
4360 ROW =  INT ((IMOD - 1) / 6)
4370 COL = IMOD - 6 * ROW - 1
4380 C1 =  INT (I / 100)
4390 C2 = I - 100 * C1
4400 C2 =  INT (C2 / 10)
4410 C3 = I - 10 *  INT (I / 10)
4420  POKE 232,NLO: POKE 233,NHI
4430 C1 = C1 + 2:C2 = C2 + 2:C3 =
     C3 + 2
4440  IF C1 = 2 THEN 4460
4450  DRAW C1 AT 45 * COL + 5,30 *
     ROW + 7
4460  IF C2 = 2 AND C1 = 2 THEN 4
     480
4470  DRAW C2 AT 45 * COL + 10,30
      * ROW + 7
4480  DRAW C3 AT 45 * COL + 15,30
      * ROW + 7
4500  POKE 232,ALO: POKE 233,AHI
4510  DRAW I AT 45 * COL + 30,30 *
     ROW + 15
4520  NEXT I
4530  GET KEY$: TEXT : HOME : GOTO
     1310
```

Auto-Run-Save, Y-t Plotter, Canary for the PET

A potpourri of programs is prasented for the PET. These include two obviously useful utility programs end one program of dubious utility.

Werner Kolbe
Herdstr. 77 CH 54 32
Neuenhof Switzerlend

When you have several programs on a tape, you can only select a specific one by entering LOAD together with the program name, and then you have to wait until the program has been loaded before you can enter RUN. This was one reason for me to develop the Auto-Run-Saver which allows you to save programs in a form so that they run automatically after the load. The second reason was, that the Auto-Run-Saver can also be used for nearly perfect program protection. If the stop-key is disabled and other possibilities of program interruption are avoided, your program cannot be stopped and therefore can not be changed nor can it be listed. Auto-Run-Saver is written for 8K PETs with the old ROMs.

Using the Program

Auto-Run-Saver mainly consists of machine code which is combined with a short BASIC loader that gives the instructions. After running the machine code is located in the last page of the 8K memory. You load the program that you want to save and place an empty tape into the cassette unit. Instead of SAVE you now enter SYS7636 and your program is saved with the auto-run feature.

Program Description

The trick of Auto-Run-Saver is, that it writes a header on the tape

```
AUTO-RUN-SAVER
5 POKE134,250:POKE135,30:CLR
10 FORI=7936TO8131:READN:POKEI,N:NEXT
20 INPUT"▒PROGRAM NAME ";A$
30 A=LEN(A$):IFA>16THENA=16
40 FORI=1TOA:POKE8057+I,ASC(MID$(A$,I,1))
50 NEXT
60 PRINT:PRINT"▒1. LOAD THE PROGRAM "A$
70 PRINT"▒2. PLACE A BLANK TAPE INTO THE CASSETTE    UNIT
80 PRINT"▒3. ENTER SYS7636
90 PRINT"▒▒FOR FURTHER RECORDS REPEAT FROM STEP 2
500 DATA165,124,141,140, 31,165,125,141,145, 31,169, 1,133,241,169,122,133
505 DATA249,169, 31,133,250,169, 75,133,238,169, 13,133,247,169, 2,133,248
510 DATA169, 22,133,229,169, 2,133,230, 32,103,246, 32,113,248,169, 1, 32
515 DATA237,245,169,111,133,247,169, 31,133,248,169,122,133,229,169, 31,133
520 DATA230, 32, 13,247, 32, 96, 31,169, 0,133,247,169, 4,133,248,165,124
525 DATA133,229,165,125,133,230, 76, 13,247, 0, 0,169,112,141, 5, 2,173
530 DATA 5, 2, 16,251, 96, 0, 0, 0, 0, 8, 0,147, 83,217, 54, 53, 54
535 DATA 13,147, 0, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
540 DATA 32,32, 0,169,169,141,125, 2,169, 5,141,126, 2,169, 0,141,123
545 DATA 2,169, 4,141,124, 2,162, 8,189,190, 2,157, 12, 2,202,208,247
550 DATA162, 8,169, 32,157,255,127,202, 16,250,142, 16,232, 76,195,243, 4
555 DATA 0,147, 82,213, 13, 0, 0, 0, 0
READY.
```

which later advises PET to load directly into its keyboard buffer (dec. 525 to 536). In our case a SYS656 together with a carriage return is entered, leading to a small machine code routine which is a part of the program name. This routine enters RUN, Ret. into the keyboard buffer, puts the correct load addresses into the according places of the cassette buffer, disables the stop-key and finally jumps to the load-routine located in the ROM at F3C3. The disassembly (listing 2) may serve to understand the whole process more in detail.

The program mainly consists of two parts. The first one from 1F00 to 1F79 does the SAVE. The second one from 1F8B to 1FBF is saved as a part of the program name and performs the LOAD and RUN of the BASIC program.

First (1F00 to 1F09) the LOAD routine is updated with the actual "End of BASIC pointer". Then all necessary pointers are set to write a header on tape. The name for the header is 75 characters long (hex. 4B) starting at 1F7A. The start address in the header is set to 020D and the end address is 0216.

After having written the header, the pointers are prepared to write a pseudo program on tape, which starts at 1F6F and ends at 1F79. This "program" will make PET assume that 8 Keys were pressed during the loading: CLR, S, y, 6, 5, CR, CLR. The subroutine 1F60 is a waiting loop in order to provide a gap between the pseudo program and the BASIC program. After writing the BASIC code on tape (1F4B to 1F5D) the "Auto-Run-Save" is complete.

When loading such a program, PET will immediately execute the SYS656, which will lead it to the code located in the disassembly listing at 1F8B. This routine prepares the pointers to load the BASIC program without header, it stores CLR, R, u, CR into the keyboard buffer and it disables the STOP-key during the loading by storing FF into E810

Protecting a Program

The Auto-Run-Saver disables the stop-key only during the loading. Therefore your BASIC program must contain the line

0 POKE 537, 136

in order to disable the stop-key during the run. Further on all the INPUT

AUTO RUN SAVER

DISASSEMBLY

```
C*    PC  SR AC XR YR SP   NV*BDIZC
.;  70ED 20 53 41 56 FE  00100000
.
.:  1F00 A5 7C        LDA  $7C
.:  1F02 8D 8C 1F     STA  $1F8C
.:  1F05 A5 7D        LDA  $7D
.:  1F07 8D 91 1F     STA  $1F91
.:  1F0A A9 01        LDA  =$01
.:  1F0C 85 F1        STA  $F1
.:  1F0E A9 7A        LDA  =$7A
.:  1F10 85 F9        STA  $F9
.:  1F12 A9 1F        LDA  =$1F
.:  1F14 85 FA        STA  $FA
.:  1F16 A9 4B        LDA  =$4B
.:  1F18 85 EE        STA  $EE
.:  1F1A A9 0D        LDA  =$0D
.:  1F1C 85 F7        STA  $F7
.:  1F1E A9 02        LDA  =$02
.:  1F20 85 F8        STA  $F8
.:  1F22 A9 16        LDA  =$16
.:  1F24 85 E5        STA  $E5
.:  1F26 A9 02        LDA  =$02
.:  1F28 85 E6        STA  $E6
.:  1F2A 20 67 F6     JSR  $F667
.:  1F2D 20 71 F8     JSR  $F871
.:  1F30 A9 01        LDA  =$01
.
.
.:  1F32 20 ED F5     JSR  $F5ED
.:  1F35 A9 6F        LDA  =$6F
.:  1F37 85 F7        STA  $F7
.:  1F39 A9 1F        LDA  =$1F
.:  1F3B 85 F8        STA  $F8
.:  1F3D A9 7A        LDA  =$7A
.:  1F3F 85 E5        STA  $E5
.:  1F41 A9 1F        LDA  =$1F
.:  1F43 85 E6        STA  $E6
.:  1F45 20 0D F7     JSR  $F70D
.:  1F48 20 60 1F     JSR  $1F60
.:  1F4B A9 00        LDA  =$00
.:  1F4D 85 F7        STA  $F7
.:  1F4F A9 04        LDA  =$04
.:  1F51 85 F8        STA  $F8
.:  1F53 A5 7C        LDA  $7C
.:  1F55 85 E5        STA  $E5
.:  1F57 A5 7D        LDA  $7D
.:  1F59 85 E6        STA  $E6
.:  1F5B 4C 0D F7     JMP  $F70D
.:  1F5E 00           BRK
.:  1F5F 00           BRK
.:  1F60 A9 70        LDA  =$70
.
.:  1F62 8D 05 02     STA  $0205
.:  1F65 AD 05 02     LDA  $0205
.:  1F68 10 FB        BPL  $1F65
.:  1F6A 60           RTS
```

statements must be replaced by an appropriate subroutine using GET A$. Of course it will still be possible to copy your program, i.e. with a second cassette recorder, but it will be quite difficult to change it in order to take out your copyright label.

Important Memory Locations:

7C, 7D	End of Basic Pointer
F1	Current device
F9, FA	Start of program name
EE	Number of characters in name
F7, F8	Pointer to program start
E5, E6	Pointer to program end
027B, 027C	Start address for load
E810	To disable stop during load, store a number higher than 9 in the low 4 bits
F5ED	Writes a header
F70D	Writes without header from addresses in F7, F8, E5, E6
F3C3	Loads program without header
F667	Sets buffer pointer
F871	Tests if cassette motor runs

Using the PET Printer 2022 as a Y-t Plotter

The Pet printer 2022 can easily be turned into a Y-t plotter using the following short program.

Listing

The function to be plotted must have the form Y = F(T). The value of Y should be calculated in a subroutine starting at line 500. Y must be between 0 and 480.

Program Description

After opening all necessary channels the line feed distance is reduced by printing CHR$(18) to channel 5. Then from line 10 to line 35 seven consecutive values of the function are calculated. The corresponding printing positions are stored in D %; the column in the printing position is stored in S%(I) where I contains the row position. The following loops from 40 to 100 determine the values of the characters that have to be transmitted to channel 5 in order to program the programmable character. For this purpose all values having the same printing position are combined. The positions which were combined are marked with D%(J) = 99. Because it is not possible to program more

```
.:  1F6B 00         BRK
.:  1F6C 00         BRK
.:  1F6D 00         BRK
.:  1F6E 00         BRK
```

Pseudo Program

```
.:  1F6F 08         PHP
.:  1F70 00         BRK
.:  1F71 93         ???
.:  1F72 53         ???
.:  1F73 D9 36 35   CMP  $3536,Y
.:  1F76 36 0D      ROL  $0D,X
.:  1F78 93         ???
.:  1F79 00         BRK
```

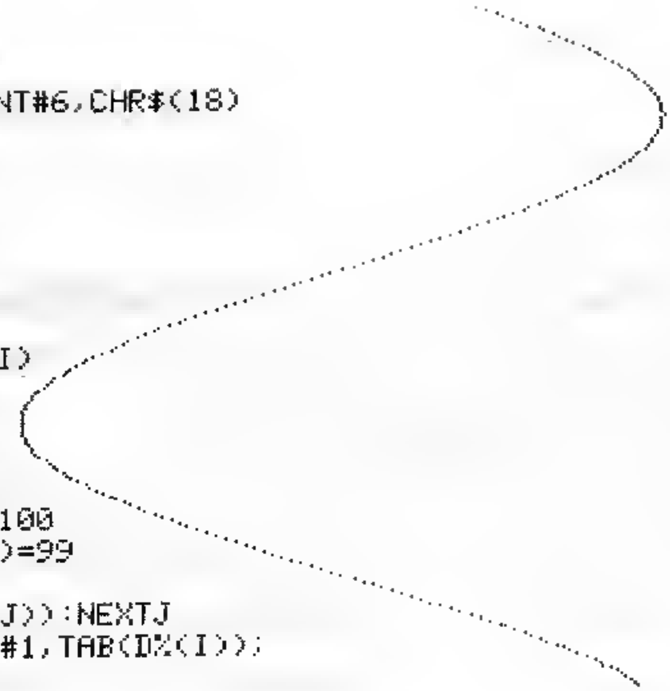
Tape Header

```
.:  1F7A 4E 41 4D   LSR  $4D41
.:  1F7D 45 4F      EOR  $4F
.:  1F7F 46 50      LSR  $50
.:  1F81 52         ???
.:  1F82 47         ???
.:  1F83 52         ???
.:  1F84 41 4D      EOR  ($4D,X)
.:  1F86 45 20      EOR  $20
.
.:  1F88 20 20 00   JSR  $0020
```

```
.:  1F8B A9 A9      LDA  =$A9
.:  1F8D 8D 7D 02   STA  $027D
.:  1F90 A9 05      LDA  =$05
.:  1F92 8D 7E 02   STA  $027E
.:  1F95 A9 00      LDA  =$00
.:  1F97 8D 7B 02   STA  $027B
.:  1F9A A9 04      LDA  =$04
.:  1F9C 8D 7C 02   STA  $027C
.:  1F9F A2 08      LDX  =$08
.:  1FA1 BD BE 02   LDA  $02BE,X
.:  1FA4 9D 0C 02   STA  $020C,X
.:  1FA7 CA         DEX
.:  1FA8 D0 F7      BNE  $1FA1
.:  1FAA A2 08      LDX  =$08
.:  1FAC A9 20      LDA  =$20
.:  1FAE 9D FF 7F   STA  $7FFF,X
.:  1FB1 CA         DEX
.:  1FB2 10 FA      BPL  $1FAE
.:  1FB4 8E 10 E8   STX  $E810
.:  1FB7 4C C3 F3   JMP  $F3C3
.:  1FBA 04         ???
.:  1FBB 00         BRK
.

.:  1FBC 93         ???
.:  1FBD 52         ???
.:  1FBE D5 0D      CMP  $0D,X
```

Y-T PLOTTER



```
 1 OPEN1,4:OPEN5,4,5:OPEN6,4,6:PRINT#6,CHR$(18)
 2 DIMA(5),D%(6),S%(6)
 3 PRINT#1,CHR$(19)
 5 DT=1
 10 FORI=0TO6:GETA$:IFA$=""THEN20
 15 PRINT#6,CHR$(24):CLOSE5:CLOSE6
 16 PRINT#1:CLOSE1:END
 20 GOSUB500:REM Y=F(T)
 30 T=T+DT:D%(I)=Y/6:S%(I)=Y-6*D%(I)
 35 NEXTI
 40 FORI=0TO6:IFD%(I)>79THEN140
 45 FORJ=0TO5:A(J)=0:NEXTJ
 50 A(S%(I))=2↑(6-I):IFI>5THEN110
 70 FORJ=I+1TO6:IFD%(I)<>D%(J)THEN100
 90 A(S%(J))=A(S%(J))+2↑(6-J):D%(J)=99
 100 NEXTJ
 110 A$="":FORJ=0TO5:A$=A$+CHR$(A(J)):NEXTJ
 115 PRINT#5,A$:IFD%(I)>0THENPRINT#1,TAB(D%(I));
 120 PRINT#1,CHR$(254)CHR$(141);
 140 NEXTI
 150 PRINT#1,CHR$(29):GOTO10
 200 REM *** YOUR FUNCTION ****
 500 Y=100+100*SIN(T/50*π/2)
 510 RETURN
READY.
```

than one character per line, every character that has to be printed in the same line must be followed by a CHR$ (141) resulting in a carriage return without line feed. The program continues to plot the function until a key is pressed.

μ

PET Singing Like a Bird

A few weeks ago my wife bought a canary. The bird was not accustomed to its new surrounding and therefore instead of singing, it sat in its cage silent and sad. Someone had to keep him company! PET could do it. The following short program turns PET into a wonderfully singing canary. You only have to connect a speaker (with a small amplifier) to the user-port output CB2.

CANARY

```
 10 PRINT"▯          ▯▯▯▯▯▯  ****   ▯CANARY▯  *****"
 20 PRINT"         ▯▯▯CONNECT A SPEAKER TO CB2.
 25 PRINT"▯▯▯           PRESS ANY KEY TO STOP"
 30 H=.5:L=51:K=136
 40 N=65:POKE59467,16:M=59464:RG=59466
 50 B=N*RND(1)+25:F=N*RND(1):A=F+B:D=(F/70+H)*RND(1)+H:Z=D*300*RND(1)/A
 60 P=A/N*H:GETA$:IFA$THENPOKE59467,0:END:STOP
 70 IFRND(1)<.1THENFORI=0TO2E3*RND(1):NEXT
 80 POKERG,L:FORI=0TOZ:IFRND(1)<HTHENPOKERG,K-L
 90 IFRND(1)>PTHEN110
 100 FORJ=ATOBSTEP-D:POKEM,J:NEXT:POKEM,0:NEXT:GOTO50
 110 FORJ=BTOASTEPD:POKEM,J:NEXT:POKEM,0:NEXT:GOTO50
READY.
```

Loading KIM-1 Tapes to AIM

Here ere the routines required to overcome the problem of loading KIM formet tapes into an AIM when the bese addresses need to be changed. They permit the user to specify from the keyboerd the new starting eddress for e loed, overriding the KIM generated sterting eddress.

Larry P. Gonzalez
Department of Physiology end Biophysics
University of Illinois Medical Center
P.O Box 6998
Chicago, IL 60680

The Rockwell AIM-65 is an excellent system for the computer hobbyist, given its ASCII Keyboard, on-line thermal printer, easy-to-use I/O chips and timers, and 8K monitor. In addition, the AIM is KIM-1 compatable and allows cassette I/O in KIM format. This means that the abundant software that is available for the KIM-1 can be read via the AIM cassette interface. This feature is particularly interesting to those of us moving up to the AIM from a KIM-1.

In actual practice, however, differences between the memory maps of the KIM-1 and the AIM-65 make the loading of KIM tapes to the AIM more difficult. The problem is in the fact that the AIM monitor makes extensive use of Page One memory locations, while the KIM-1 does not. In particular, 80 bytes of AIM Page One, beginning at location $0116, are used as the tape I/O buffer. So, although the KIM-1 can load programs Into Page One from tape, the AIM cannot; KIM tape files which load to Page One cannot be loaded with the AIM tape load routines.

The KIM monitor has a feature which allows cassette files to be loaded with a starting address different from the load address stored with the tape file. This is done by specifying "FF" as the file ID. The file name and load address on tape are ignored and the file is loaded with the starting address previously entered into RAM (KIM locations $17F5 and $17F6). Unfortunately, this feature was not included in the AIM routines that load KIM-format tapes. This problem of loading KIM-1 tapes has been noted by other authors (Burnett, 1979; Tripp, 1979), but no solutions have been presented.

The program below is a simple modification of the Rockwell AIM monitor routine to load KIM-format tapes to a new load address. Comments are included in the program, so little explanation should be required. The New Load Address is stored on Page Zero at locations $0000 and $0001. These could be changed, however, to any convenient location. The entry point to the program is at $0900. The program is completely relocatable; all that is required to relocate the program is that this entry point be changed during assembly.

The assembly-language source version as prepared on the AIM Editor is shown in Figure 1, with the assembly listing and symbol table in Figure 2, and the disassembled listing and hex dump in Figure 3.

Execute the program with the program counter set to $0900. The message "To=" will be displayed. Enter the new load address followed by a carriage return, and then continue as for a normal tape load. Don't forget to change the tape speed ($A408) to the appropriate value for your KIM-format tapes ($5A or $5B) prior to running this program.

I have found this program to be very useful in gaining access to programs which were initially dumped to tape from a KIM-1. Now I don't have to enter all my KIM programs by hand to make them available on the AIM, even if the original tape loaded into Page One.

References

Burnett, J. An AIM-65 user's notes. MICRO,1979, 12:5-7
Tripp, R.M. Ask the Doctor, Part V. MICRO, 1979 13:34-36

μ

Larry P. Gonzalez is an Assistant Professor of physiology and biophysics at the University of Illinois Medical Center. He has 12 years in the use of minicomputers for real-time data acquisition and signal analysis. During the last two years he has been developing a system using an AIM-65 in the collection and analysis of electrophysiological data.

Figure 1. **Source Listing: KIM Tape Load to New Address.**

```
*=0000
.PAGE
;KIM TAPE LOAD TO
;      NEW ADDRESS
;
;** BY L.P  GONZALEZ
;
.PAGE      'MEMORY
LOCATIONS'
NEWAD=$00;NEW ADDRES
INFLG=$A412
SAVA=$A421
ADDR=$A41C
CKSUM=$A41E
.PAGE  'SUB-ROUTINE
EQUATES'
START=$E182
CKERR=$E385
RBYTE=$E3FD
STBYTE=$E413
GETID=$E425
CHEKAR=$E54B
TO=$E7A7
FNAM=$E8A2
CRLF=$E9F0
PACK=$EA84
CLRCK=$EB4D
TAISET=$EDEA
GETTAP=$EE29
.PAGE 'MAIN PROGRAM'
.SKIP
*=$0900
;
;DISPLAY "TO="
;& READ NEW ADDRESS
;
JSR TO
;
;STORE NEW ADDRESS
;
LDA ADDR
STA NEWAD
LDA ADDR+1
STA NEWAD+1
;
;SET INPUT DEVICE
;CODE FOR KIM-1 TAPE
;
LDA #$4B
STA INFLG
LDX #00
;
;GET FILENAME AND
;      TAPE UNIT
JSR FNAM
;LOAD KIM-1 TAPE
;
LOADKI JSR CLRCK
LOADK1 JSR TAISET
LOADK2 JSR GETTAP
CMP #'*
BEQ LOADK3
CMP #$16
BNE LOADK1
BEQ LOADK2
LOADK3 JSR RBYTE
STA SAVA
;
;READ BUT IGNORE
;OLD LOAD ADDRESS
;
;REPLACE WITH NEW
;LOAD ADDRESS
;
JSR CHEKAR
JSR CHEKAR
LDA NEWAD
STA ADDR
LDA NEWAD+1
STA ADDR+1
JSR GETID
CMP SAVA
BNE LOADKI
LOADK5 LDX #$02
LOADK6 JSR GETTAP
CMP #'/
BEQ LOADK7
JSR PACK
BCC J1
JMP CKERR
J1 DEX
BNE LOADK6
JSR STBYTE
JMP LOADK5
LOADK7 JSR RBYTE
CMP CKSUM
BEQ J2
JMP CKERR
J2 JSR RBYTE
CMP CKSUM+1
BEQ J3
JMP CKERR
J3 JSR CRLF
;
;RETURN TO MONITOR
;
JMP START
.PAGE 'PROGRAM END'
.PAGE
.END
```

Figure 2. **Assembly Listing and Symbol Table : KIM Tape Load to New Address**

```
==0000
        *=0000
------------------
;KIM TAPE LOAD TO
;      NEW ADDRESS
;
;** BY L.P.  GONZALEZ
;
-------------------
MEMORY LOCATIONS

==0000 NEWAD=$00;NEW
       ADDRES
==0000 INFLG=$A412

==0000 SAVA=$A421

==0000 ADDR=$A41C

==0000 CKSUM=$A41E

-------------------
SUB-ROUTINE EQUATES

==0000 START=$E182

==0000 CKERR=$E385

==0000 RBYTE=$E3FD

==0000 STBYTE=$E413

==0000 GETID=$E425

==0000 CHEKAR=$E54B

==0000 TO=$E7A7

==0000 FNAM=$E8A2

==0000 CRLF=$E9F0

==0000 PACK=$EA84

==0000 CLRCK=$EB4D

==0000 TAISET=$EDEA

==0000 GETTAP=$EE29
```

```
--------------------
MAIN PROGRAM


==0000
         *=$0900
==0900
;
;DISPLAY "TO="
;& READ NEW ADDRESS
;
20A7E7 JSR TO
;
;STORE NEW ADDRESS
;
AD1CA4 LDA ADDR
8500   STA NEWAD
AD1DA4 LDA ADDR+1
8501   STA NEWAD+1
;
;SET INPUT DEVICE
;CODE FOR KIM-1 TAPE
;
A94B   LDA #$4B
8D12A4 STA INFLG
==0912
A200   LDX #00
;
;GET FILENAME AND
;      TAPE UNIT
;
20A2E8 JSR FNAM
;
;LOAD KIM-1 TAPE
;
==0917 LOADKI
204DEB JSR CLRCK
==091A LOADK1
20EAED JSR TAISET
==091D LOADK2
2029EE JSR GETTAP
C92A   CMP #'*
F006   BEQ LOADK3
C916   CMP #$16
D0F2   BNE LOADK1
F0F3   BEQ LOADK2
==092A LOADK3
20FDE3 JSR RBYTE
8D21A4 STA SAVA
;
;READ BUT IGNORE
;OLD LOAD ADDRESS
;
;REPLACE WITH NEW
;LOAD ADDRESS
;
204BE5 JSR CHEKAR
204BE5 JSR CHEKAR
A500   LDA NEWAD
8D1CA4 STA ADDR
==093B
A501   LDA NEWAD+1
8D1DA4 STA ADDR+1
2025E4 JSR GETID
CD21A4 CMP SAVA
D0CF   BNE LOADKI
==0948 LOADK5
A202   LDX #$02
==094A LOADK6
2029EE JSR GETTAP
C92F   CMP #'/
F011   BEQ LOADK7
2084EA JSR PACK
9003   BCC J1
4C85E3 JMP CKERR
==0959 J1
CA     DEX
D0EE   BNE LOADK6
2013E4 JSR STBYTE
4C4809 JMP LOADK5
==0962 LOADK7
20FDE3 JSR RBYTE
CD1EA4 CMP CKSUM
F003   BEQ J2
4C85E3 JMP CKERR
==096D J2
20FDE3 JSR RBYTE
CD1FA4 CMP CKSUM+1
F003   BEQ J3
4C85E3 JMP CKERR
==0978 J3
20F0E9 JSR CRLF
;
;RETURN TO MONITOR
;
4C82E1 JMP START
--------------------
PROGRAM END

--------------------

        END
 ERRORS= 0000


     SYMBOL TABLE
     ADDR    A41C
     CHEKAR  E54B
     CKERR   E385
     CKSUM   A41E
     CLRCK   EB4D
     CRLF    E9F0
     FNAM    E8A2
     GETID   E425
     GETTAP  EE29
     INFLG   A412
     J1      0959
     J2      096D
     J3      0978
     LOADK1  091A
     LOADK2  091D
     LOADK3  092A
     LOADK5  0948
     LOADK6  094A
     LOADK7  0962
     LOADKI  0917
     NEWAD   0000
     PACK    EA84
     RBYTE   E3FD
     SAVA    A421
     START   E182
     STBYTE  E413
     TAISET  EDEA
     TO      E7A7
     NEWAD   0000
     LOADKI  0917
     LOADK1  091A
     LOADK2  091D
     LOADK3  092A
     LOADK5  0948
     LOADK6  094A
     J1      0959
     LOADK7  0962
     J2      096D
     J3      0978
     INFLG   A412
     ADDR    A41C
     CKSUM   A41E
     SAVA    A421
     START   E182
     CKERR   E385
     RBYTE   E3FD
     STBYTE  E413
     GETID   E425
     CHEKAR  E54B
     TO      E7A7
     FNAM    E8A2
     CRLF    E9F0
     PACK    EA84
     CLRCK   EB4D
     TAISET  EDEA
     GETTAP  EE29
```

Figure 3. Disassembled Listing and Hex Dump: KIM Tape Load to New Address

```
<K>*=0900
/49
 0900 20 JSR E7A7
 0903 AD LDA A41C
 0906 85 STA 00
 0908 AD LDA A41D
 090B 85 STA 01
 090D A9 LDA #4B
 090F 8D STA A412
 0912 A2 LDX #00
 0914 20 JSR E8A2
 0917 20 JSR EB4D
 091A 20 JSR EDEA
 091D 20 JSR EE29
 0920 C9 CMP #2A
 0922 F0 BEQ 092A
 0924 C9 CMP #16
 0926 D0 BNE 091A
 0928 F0 BEQ 091D
 092A 20 JSR E3FD
 092D 8D STA A421
 0930 20 JSR E54B
 0933 20 JSR E54B
 0936 A5 LDA 00
 0938 8D STA A41C
 093B A5 LDA 01
 093D 8D STA A41D
 0940 20 JSR E425
 0943 CD CMP A421
 0946 D0 BNE 0917
 0948 A2 LDX #02
 094A 20 JSR EE29
 094D C9 CMP #2F
 094F F0 BEQ 0962
 0951 20 JSR EA84
 0954 90 BCC 0959
 0956 4C JMP E385
 0959 CA DEX
 095A D0 BNE 094A
 095C 20 JSR E413
 095F 4C JMP 0948
 0962 20 JSR E3FD
 0965 CD CMP A41E
 0968 F0 BEQ 096D
 096A 4C JMP E385
 096D 20 JSR E3FD
 0970 CD CMP A41F
 0973 F0 BEQ 0978
 0975 4C JMP E385
 0978 20 JSR E9F0
 097B 4C JMP E182
< > 0900 20 A7 E7 AD
< > 0904 1C A4 85 00
< > 0908 AD 1D A4 85
< > 090C 01 A9 4B 8D
< > 0910 12 A4 A2 00
< > 0914 20 A2 E8 20
< > 0918 4D EB 20 EA
< > 091C ED 20 29 EE
< > 0920 C9 2A F0 06
< > 0924 C9 16 D0 F2
< > 0928 F0 F3 20 FD
< > 092C E3 8D 21 A4
< > 0930 20 4B E5 20
< > 0934 4B E5 A5 00
< > 0938 8D 1C A4 A5
< > 093C 01 8D 1D A4
< > 0940 20 25 E4 CD
< > 0944 21 A4 D0 CF
< > 0948 A2 02 20 29
< > 094C EE C9 2F F0
< > 0950 11 20 84 EA
< > 0954 90 03 4C 85
< > 0958 E3 CA D0 EE
< > 095C 20 13 E4 4C
< > 0960 48 09 20 FD
< > 0964 E3 CD 1E A4
< > 0968 F0 03 4C 85
< > 096C E3 20 FD E3
< > 0970 CD 1F A4 F0
< > 0974 03 4C 85 E3
< > 0978 20 F0 E9 4C
< > 097C 82 E1 00 00
```

K I M A I M S Y M T I M

END FRUSTRATION!!

FROM CASSETTE FAILURES
PERRY PERIPHERALS HAS
THE HOE SOLUTION

OMNIOISK SYSTEMS (5" and 8")

ACCLAIMED HDE SOFTWARE

- Assembler, Oynamic Debugging Tool, Text Output Processor, Comprehensive Memory Test
- Coming Soon—HOE BASIC

PERRY PERIPHERALS S-100 PACKAGE

Adds Omnidisk (S") to Your KIM/S-100 System

- Construction Manual—No Parts
- FODS & TEO Diskette
- $20. +$2. postage & handling. (NY residents add 7% tax) (specify for 1 or 2 drive system)

Place your order with:
PERRY PERIPHERALS
P.O. Box 924
Miller Place, N.Y. 11764
(516) 744-6462

Your Full-Line HDE Distributor/Exporter

Frustrated? One-board Micro-C's too expensive, Insufficient I/O lines for full control, Generat purpose Micro an overkill, Fixed power supply requirements, costly development systems, if available, lack ot user flexibility?

The MICROsport MicroComputer (MMC), In addition to being a complete microcomputer on a 4½" X 6½" pc board, is the nucleus ot a full hardware/software development system. Sottware can be developed for the MMC using any 6502-based computer. The In-Circuit Emulator (ICE) permits full MMC sottware/hardware debugging, then adding the EPROM Programmer Adaptor, any single +5V EPROM, such as the Intel-type 2716 or 2758, can be programmed without additional equipment. The MMC is the ideal dedicated controller tor use in control/monitor systems, laboratory experiments, timing, Intelligent interfaces, security systems, and other applications requiring a low cost controller.

Features

CPU: MPS6503, **RAM:** 1 Kbytes, **EPROM:** Socket tor Intel-type 2716 or 2758. (User supplied), **I/O:** 2 MPS6522 VIAs: 32 I/O lines, 8 edge detector/control lines, 4-16 bit counters, 8 user-detined pins. **Serial:** 20 mA loop on-board, **Interrupts:** Power-on and manual reset, non-maskable and maskable, **Power:** +5V regulated or 9—20V unregulated AC or DC, **All** ICs socketed, LED power on Indicator, CMOS-RAM w/battery back-up optional. **Options:** In-Circuit Emulator; EPROM Programming Adaptor; MMC Development Model MMC/03D with zero Insertion force sockets (3); EPROM Programming Services; and application software development.

Prices: Kits from $89.00 Application units from $119.00 **MMC/03S:** Complete development system Including MMC/03D; MMC/03ICE; MMC/03EPA; and software $250.00 (otfer good until Dec. 31, 1980). Checks or money orders; No C.O.D..'s. For further info write or call:

R.J. Brachman Associates, Inc.
P.O. Box 1077
Havertown, PA 19083
(215) 622-5495

Decision Systems

*Presenting the Other Side of the Apple II**

INDEXEO FILES

ISAM-DS is an intagrated sel of routines for the craation end manipulatron ot indexed files. **ISAM-DS** providas cepabilities comperable lo those on large meinframes. You can repidly retrieve racords by kay value or partial key value (retrieves any record in a 200 record lile, 60 char/record, in less then 3 seconds compared lo e maximum ot 38 seconds lor e DOS sequential filel. Files navar have lo be reorganized. Duplicale key values mey ba used. Records may elso be retrieved in sequance. **ISAM-DS** roulinas ara easily integrated inlo Applesoft programs — they use lass then **3K RAM** plus en index table.

Requires: Disk, Applesoft (32K ROM or 48K RAM)

$50

STRUCTURED BASIC

PBASIC-DS is a sophisticelad preprocessor for structured **BASIC**. Now you can gain the power of **PASCAL**-like logic structures el a traction of the cosl. Use all ragular **BASIC** slatemenls plus 14 commands and 11 new slatements/structures (**WHILE, UNTIL, CASE,** etc.). **PBASIC-DS** cen be used lo develop **INTEGER** or **APPLESOFT** programs. It is e greel wey lo learn and use structured logic concepls

Requires: Disk, Applesoft (32K ROM or 48K RAM)

$35

(Texas residents add 5% tax)

Decision Systems
P.O. Box 13006
Denton, TX 76203

*Apple II is e registered trademark of the Apple Computer Co.

IMPORTANT NEW PRODUCT

CONTINUOUS PRINTOUT PAPER

MINI-PAK

- Clean, carbonless paper
- Finesl quality blank while paper with ½" detachable marginal holes bolh sides, leaving a standard 8½ x 11 sheel
- Duplicale and Iriplicate forms make clear copies with no carbon required.
- Ideal for letters, reports, work copies.
- Available in one, two or three plys.
- Lightweight—only 14 pounds!
- Packed 1500 sheets for 1 part, 750 sheets-2 part and 500 sheets-3 part.

$25.00 per carton!*

REGARDLESS OF HOW MANY PLYS

TO ORDER SEND CHECK OR MONEY ORDER

VISA AND M/C ACCEPTEO—SEND ACCOUNT NUMBER, EXPIRATION DATE WITH ORDER

*ADD $5.00 FOR UPS AND HANDLING CHARGES
(MASS RESIDENTS ADD 5% SALES TAX)

COMPUTER FORMS CO.
34 CHELMSFORD STREET
CHELMSFORD, MA 01824

(DEALER INQUIRIES INVITED)

Compact

Another member of the "Stripper" family - programs to strip REMarks from BASIC programs - this version works on the AIM and does the stripping in place. It does not require the use of disk or cassette tapes.

Steve Bresson
1302 Strawberry Ln
Hanover, MD 21076

The "Apple Stripper" Program in MICRO 23:11-12 removes the REM statements from a program using a BASIC program and a disk file. I would love to use this method, but a 4K AIM-65 with a tape recorder would make for a long wait. The assembly program given here was programmed slightly before "Apple Stripper", and does the compaction in place.

Compact is a program to strip out blanks and REMs from a BASIC program. This is done to save space and increase the operating speed of a large or heavily documented program.

The program is run as follows:

1) Load in COMPACT. I put it in high memory for the 4K AIM.
2) Initialize BASIC. Make sure it does not overlap COMPACT.
3) Escape back to the Monitor.
4) Run COMPACT. On the AIM, just hit <FL>.

COMPACT operates by scanning through the BASIC program looking for quotes, blanks, and REM tokens. Blanks are stripped out as they are encountered. All text between quotes is ignored. A REM forces one of two things to be done. If the character counter is zero, then the REM is at the beginning of the line and the whole line is removed. A non-zero character count indicates the REM is placed after text, so only the remainder of the line is removed. In all cases, pointers to the locations to be removed are passed to subroutine PACK, which does the actual deletion.

PACK performs the nontrivial task of closing up the BASIC program to overwrite the unwanted string. Then the BASIC pointers are changed so that BASIC still knows where the program is located.

The final operation in PACK is a jump to $B329. This is a subroutine in the BASIC ROM which relinks the line pointers of the program. The NBLP (New BASIC Line Pointer) subroutine expects the "standard" BASIC line format of:

```
 0 1 2  3   4.................n
:lo, hi:lo, hi:basic text.........:00:
LINE     LINE          END OF
PTR      NUMBER        LINE
```

It scans through each line, first checking the line Pointer high byte for a $00, which would indicate the end of the program. If the line pointer is not zero, the line is scanned until a $00 is found. That address plus one is the beginning of the next line and is placed in the line pointer. The NBLP pointer is moved to the beginning of the new line and the process starts over.

For those of you who do not have the AIM, an assembly language NBLP is also listed. Assuming your BASIC stores its programs in the same format as the AIM, only a couple of things need be known to make this program run on your machine:

1) The address of the Beginning of BASIC (BOB) pointer.
2) The address of the Top of BASIC (TOB) pointer.
3) A couple of 2 byte locations in page 0 for temporary use as pointers.

By plugging these values into the listing you should not (hopefully) have any problems.

Program Listing

1) Assembler output of "COMPACT".
2) Start up of BASIC so that top of memory is not affected.
3) Crossed out PGM, skip this.
4) BASIC PGM to be compacted.
5) Test run to show program output.
6) List of BASIC Pointers at $0075 — top of BASIC before compaction.
7) <[> <F1> KEY RUN COMPACT
8) RUN of Compacted PGM.
9) List of Compacted PGM.
10) New top of BASIC PTR =$0261, OLD=$02 LAC!
11) Change of M.L. PGM to use 'NBLP1' instead of BASIC

ROM PGM.

12) BASIC test PGM with additional lines.

13) Run of COMPACT with 'NBLP1'

14) New Listing.

μ

```
ERRORS= 0000
4 3/80
(1) PASS 1
PASS 2
--------------------
COMPACT

*=$0000
; 10 3.11.80. SLB
 COMPACT A BASIC PGM
 IN MEMORY
--------------------

 STEVE BRESSON
 1302 STRAWBERRY LN
 HANOVER, MD 21076
--------------------

==0000 BOB=$73

==0000 TOB=$75

==0000 DEL=$A7

==0000 [illegible]=$5A

==0000 STO=$6C

==0000 NBLP=$B329

==0000 LINE=$FE

==0000 TMP=$FC

==0000 FLAG=$FB

==0000 REM=$8E

==0000 QFLG=$FA

==0000
        *=$10C
==010C
4C C00E JMP MAIN
==010F
        *=$EC0
--------------------

==0EC0 MAIN
A573   LDA BOB
85FE   STA LINE
A574   LDA BOB+1
85FF   STA LINE+1
4CCF0E JMP M1
==0ECB M0
C8     INY
==0ECC MA
20380F JSR ADYLL
==0ECF M1
A000   LDY #0
84FB   STY FLAG
84FA   STY QFLG
C8     INY
B1FE   LDA (LINE),Y
D001   BNE *+3
60     RTS
A004   LDY #4
==0EDD M2
B1FE   LDA (LINE),Y
F0EA   BEQ M0
==0EE1 M3
204A0F JSR QUOTE
1006   BPL M4
==0EE6 M3A
E6FB   INC FLAG
C8     INY
D0F2   BNE M2
00     BRK
==0EEC M4
C98E   CMP #REM
F015   BEQ M6
C920   CMP #' '
D0F2   BNE M3A
20280F JSR ADYLD
==0EF7 M5
C8     INY
B1FE   LDA (LINE),Y
F0FB   BEQ M5
==0EFC M5A
20300F JSR ADYLS
20620F JSR PACK
4CCF0E JMP M1
==0F05 M6
A5FB   LDA FLAG
D006   BNE M6A
20590F JSR LTD
4C170F JMP M6B
==0F0F M6A
20290F JSR ADYLD
88     DEY
A900   LDA #0
91FE   STA (LINE),Y
==0F17 M6B
A000   LDY #0
B1FE   LDA (LINE),Y
856A   STA SAV
C8     INY
B1FE   LDA (LINE),Y
856B   STA SAV+1
20620F JSR PACK
4CCF0E JMP M1
--------------------

==0F28 ADYLD
20400F JSR ADYL
85A8   STA DEL+1
86A7   STX DEL
60     RTS
==0F30 ADYLS
20400F JSR ADYL
856B   STA SAV+1
866A   STX SAV
60     RTS
==0F38 ADYLL
20400F JSR ADYL
85FF   STA LINE+1
86FE   STX LINE
60     RTS
==0F40 ADYL
18     CLC
==0F41 ADYL1
98     TYA
65FE   ADC LINE
AA     TAX
A5FF   LDA LINE+1
6900   ADC #0
60     RTS

==0F4A QUOTE
C922   CMP #'"'
D008   BNE Q2
48     PHA
A5FA   LDA QFLG
4980   EOR #$80
85FA   STA QFLG
68     PLA
==0F56 Q2
24FA   BIT QFLG
60     RTS

==0F59 LTD
A6FE   LDX LINE
86A7   STX DEL
A6FF   LDX LINE+1
86A8   STX DEL+1
60     RTS
--------------------
```

```
;PACK V1C.2.25.80.SL
B
;DEL:HOLDS DEL AREA
START ADDR
;SAV: HOLDS END OF D
EL AREA+1 ADDR
==0F62 PACK
A5A8    LDA DEL+1
856D    STA GTO+1
A5A7    LDA DEL
38      SEC
E56A    SBC SAV
18      CLC
6575    ADC TOB
A675    LDX TOB
866A    STX SAV
==0F72
8575    STA TOB
856C    STA GTO
A576    LDA TOB+1
69FF    ADC #$FF
8576    STA TOB+1
E5A8    SBC DEL+1
AA      TAX
38      SEC
A5A7    LDA DEL
==0F82
E575    SBC TOB
A8      TAY
B003    BCS PK1
E8      INX
C66D    DEC GTO+1
==0F8A PK1
18      CLC
656A    ADC SAV
9003    BCC PK2
C66B    DEC SAV+1
18      CLC
==0F92 PK2
B16A    LDA (SAV),Y
916C    STA (GTO),Y
C8      INY
D0F9    BNE PK2
E66B    INC SAV+1
E66D    INC GTO+1
CA      DEX
D0F2    BNE PK2
;JMP NEW BASIC LINE
PTR SUBR
4C29B3 JMP NBLP
---------------- ---

==0FA3
;REPLACEMENT FOR THE
 RIM BASIC ROM PGM
==0FA3 NBLP1
A473    LDY BOB
A574    LDA BOB+1
84FC    STY TMP
85FD    STA TMP+1
18      CLC
==0FAC NBLP2
A001    LDY #1
;END OF PGM?
B1FC    LDA (TMP),Y
F01D    BEQ NBLP4
A004    LDY #4
;NO! LOOK FOR $00
==0FB4 NBLP3
C8      INY
B1FC    LDA (TMP),Y
D0FB    BNE NBLP3
C8      INY
;FOUND IT!
98      TYA
;CALC ADDRESS
65FC    ADC TMP
AA      TAX
A000    LDY #0
91FC    STA (TMP),Y
 CHANGE BASIC PTR
A5FD    LDA TMP+1
==0FC4
6900    ADC #0
C8      INY
91FC    STA (TMP),Y
;CHANGE BEGINNING OF
 LINE PTR
86FC    STX TMP
85FD    STA TMP+1
90DD    BCC NBLP2
==0FCF NBLP4
60      RTS
-------------------

==0FD0 THEEND
         END
 ERRORS= 0000
```

(2)
```
<5>
MEMORY SIZE? 3750
WIDTH?
 3220 BYTES FREE
  AIM 65 BASIC V1.1
<M>=7B  A6 0E 1A 00
```

(4)
```
LIST
 10 REM TEST OF COMP
ACT OF 4/2/80
 20 PRINT "THIS IS A
TEST "
 30 A = 5 : B = 7
 40 C = 11 : REM ABC
D
 50 A = 6 : PRINT "
DONE! " : REM XXXXX
 60 GOTO 70 : REM YY
YY
 70 END
```

(5)
```
RUN
THIS IS A TEST
 DONE!
```

(6)

Top of BASIC

```
<M>=0075 AC 02 C1 02
< > 0079 C1 02 A6 0E
```

(7)
```
<L>
JUST RAN THE TEST
```

(8)
```
RUN
THIS IS A TEST
 DONE!
```

(9)
```
LIST
 20 PRINT"THIS IS A
TEST "
 30 A=5:B=7
 40 C=11
 50 A=6:PRINT" DONE!
"
 60 GOTO70
 70 END
```

Top of BASIC

(10)
```
<M>=0075 61 02 76 02
< > 0079 76 02 A6 0E
```

(11)

JMP NBLPI

```
<M>=FA0  4C A3 0F A4
TEST WITH NBLP1 LINK
ED TO ASM PGM
```

(12)
```
RUN
THIS IS A TEST
 DONE!

LIST
 10 REM TEST2 !!!
 20 PRINT"THIS IS A
TEST "
```

```
 30 A=5:B=7
 40 C=11
 45 D = 55:REM ASDDF
 50 A=6:PRINT" DONE!
 "
 60 GOTO70
 65 D = C : PRINT"HE
LP!" : REM XXXY
 70 END

RUN
THIS IS A TEST
 DONE!
 DONE!
```
(13)
```
[illegible]
[illegible]
RUN
THIS IS A TEST
 DONE!
```
(14)
```
LIST
 20 PRINT"THIS IS A
TEST "
 30 A=5:B=7
 40 C=11
 45 D=55
 50 A=6:PRINT" DONE
 "
 60 GOTO70
 65 D=C:PRINT"HELP!"
 70 END
```

Steve Bresson is a 1977 graduate of the University of Akron with a B.S.E.E. He currently works for the Dept. of Defence in Baltimore, Maryland. He has experience in Fortran, APL, CHPL, 8080, Z80, and 6502.

Steve owns an AIM-65 and has many plans for it, but hasn't gotten around to building any of them yet.

A C1P and H14 System, Part 2

A previous erticle provided the informetion required to interface en H14 printer to an OSI C1P computer. This erticle provides the softwere necessery to drive the printer.

William L. Taylor
246 Flora Rd.
Leavittsburg, OH 44430

In a previous part of this series of articles I promised some software to further the use of the C1P and a printer. In my system the printer is a Heath H14. Yours may be of another manufacture. In any case, this software should support your printer if you have used the modifications to your C1P and have interfaced your printer. This program will help you with your task of writing all forms of business and personal letters.

The program in listing 2 gives the user of the C1P and the H14 system the needed software to allow the format of business letters. This program will allow the user to develop letters, which are in the modified block form. The program allows the user to store the heading; the complimentary close; and the identification as a permanent part of the program. That is, your street address in this heading, the closing, compliment such as "Sincerely yours" and your name as the identification. The inside address; the salutation; and the body of the letter are entered on query from the computer.

To begin, the program at line 30 through 65 is used to develop the heading; the inside address, and the salutation of the letter being written. In the example program, lines 30 and 35 contain the heading. This heading is stored in Strings and is a permanent part of the program. You will have to enter your own address in these two lines. This data will be printed out when you call for a printout of your letter. At lines 37 and 40, you will be asked for the month which will be stored in A$(4). Line 40 gets the date and year. The date and year is stored in the numerical varibles E and Y. Lines 45 through 55 are used to collect the information for the inside address and the salutation. This data is stored in Strings. These Strings are: A$(9), A$(5), A$(6), A$(7), and A$(8). These Strings are not a permanant part of the program. That is, each time the program is run these strings will require new data and must be input by the user. These are all input statements. Lines 60 and 65 form the complimentary close and the identification for the letter being composed. This data is permanant and will have to be entered when you load the program for the first time. To continue, the program at lines 70 through 210 is where the body of the letter is entered by the user. This data or letter text is stored in String arrays. Up to 256 lines of text can be entered and stored in memory arrays. B$(I) holds each line of text. That is, as you type in each line of text, that line will be placed in B$(I).

The variable I contains the line number for the text data whch goes into B$. if I equaled 1 then B$(1) would become position B$(1) exc. The length of each line in the text for the body of the letter is set to a maximum of characters. If you type more characters in the line than the set length the computer responds with overwidth and the line of text is deleted. You will have to type in the line again. The statement at line 140 sets down a pointer to indicate where a line will end. This pointer should not be exceeded. At line 180, line 180 is the INPUT statement for the text input if all the letter text has been completed. Line 210 causes a RETURN through the body routine if the letter text has not been completed. When the body of the letter has been completed, and the user types the escape key(&) the program branches to line 5000. The routine at lines 5000 through 6000 is used to insure that the letter is placed correctly on the page. This subroutine checks for the number of lines that the user has entered into memory. The body of the text is read and the number of text lines are stored in the variable L, the variable L is checked against a constant of 32. The value of variable L is subtracted from 32 and stored in the X variable. The X variable is then divided by 3.

The final value of X is used to space the letter properly on the page. That is, the paper will be advanced the amount that is equal to ⅓ X. For example, if you only had 6 lines of text in the body of your letter, this value would be subtracted from 32. The X variable then would be 26. After dividing the X Variable by 3, X would be approximately 8. This value will advance the paper 8 spaces before the heading and date are printed

out. The routine from 5000 to 5070 obtains the final value for the X variable. The routine from 5080 through 5095 generates the line feeds for the paper advance. This is accomplished with a PRINT statement in a FOR-NEXT loop. At line 6000 a RETURN is executed and the program returns to line 1000.

Beginning at line 1080 the main body of the letter text is retrieved from the array and printed out to the screen and the printer. This is done with the FOR-NEXT loop at lines 1080 and 2000. At line 2007 a gosub is executed. The subroutine at 4000 is used to produce the correct amount of spaces between the body of the letter and the complimentary close. This subroutine uses the value in the X variable in the same manner as the routine at line 5000. At this point, FI should explain the statement at line 4000. The statement at line 4000 uses the keyword LOAD followed by the keyword POKE, 515,0. The statement LOAD: POKE 515,0 actually returns the C1P to the fast CRT routine. The LOAD command expects an INPUT from either the cassette recorder or the keyboard, but immediately we turn off the LOAD command by POKEing the flag at 515 with zero. This disables the LOAD command and returns the program to line 4030.

On return from the subroutine at 4070 the complimentary close and identification are printed in the letter. At line 2033 we again return the program from a SAVE mode to the regular program execution with the statement LOAD: POKE 515,0. From this point the program jumps to line 3000 where the user will be asked if more copies of the letter are desired. The "Letter Writer" program has some features that are hidden from the quick observer. The main feature is that the text editing feature of the C1P's ROM BASIC can be used to edit the text when entering the lines in your letter. This is done with the use of Control C and Control P. If a letter in your text was incorrectly inserted, you may change the letter by typing a control O. This will delete the last letter that you entered. Also, if a complete word was mispelled simply count the letters in the word and type Control O the correct number of times that were in the word. Now type in the correct word or correct spelling

```
OK
LIST

 1 REM LETTER WRITER BY W. L. TAYLOR
 2 REM AUGUST 15,1979
 3 PRINT"      LETTER WRITER  "
 4 PRINT:PRINT:PRINT
 10 PRINT" DATE YEAR AND LETTER TEXT MUST BE
                                        ENTERED"
 30 A$(2)="246 Flora Road"
 35 A$(3)="Leavittsburg, Ohio"
 37 INPUT"MONTH";A$(4)
 40 INPUT" TODAYS DATE---AND YEAR";E,Y
 45 INPUT"COMPANY";A$(5)
 47 INPUT"STREET ADDRESS";A$(6)
 49 INPUT" CITY,STATE ZIP";A$(7)
 50 INPUT" PERSON";A$(8)
 55 INPUT" GREETING";A$(9)
 60 A$(10)="Sincerely,"
 65 A$(1)="Mr. William L. Taylor"
 70 D=64
 80 I=256
 90 DIM B$(I)
 100 PRINT
 110 FOR I=1 TO 256
 120 PRINT I
 140 POKE 54181+(D-50),94
 180 INPUT B$(I)
 190 IF LEN((B$(I)))>D THEN PRINT"OVERWIDTH";I=I-1
 200 IF B$(I)=">" THEN 5000
 210 NEXT I
 250 GOTO 5000
 1000 SAVE
 1005 PRINT TAB(50);A$(2)
 1010 PRINT TAB(50);A$(3)
 1015 PRINT TAB(50);A$(4);E;Y
 1020 PRINT:PRINT:PRINT:PRINT
 1030 PRINTA$(5)
 1035 PRINTA$(6)
 1040 PRINTA$(7)
 1050 PRINT:PRINT
 1055 PRINTA$(8)
 1060 PRINT:PRINT
 1070 PRINTA$(9)
 1075 PRINT:PRINT
 1080 FOR J=1 TO I-1
 1090 PRINTB$(J)
 2000 NEXT J
 2007 GOSUB 4000
 2010 PRINT
 2020 PRINT TAB(50);A$(10)
 2025 PRINT:PRINT:PRINT
 2030 PRINT TAB(50);A$(1)
 2035 LOAD: POKE 515,0
```

for the word that was mispelled. If a complete line in the letter were needed, you simply type a Control P. This will delete the entire line of text. This program also allows the use of the C1P's lower case letter feature. That is, when you wish to enter lower case letters you need only to release the Shift-Lock key to shift into lower case letters mode. This will allow you to use both capital and lower case letters in your text.

In part one of this series I gave the reader the necessary hardware information to allow the C1P to function in a 300 Baud RS232C mode. The use of the C1P and a Heath H14 Printer was described along with the modifications to the printer to be used with the Challenger C1P. Some software was given. This article has been an extension of that article. I hope that this series has been of interest and will be a tool to help you further improve your computer system and software.

μ

```
2040 GOTO 3000
3000 PRINT" DO YOU WANT ANOTHER COPY.
                              TYPE YES OR NO"
3010 INPUT Q$
3020 IF Q$="YES" THEN GOTO 5000
3030 END
4000 LOAD:POKE 515,0
4030 FOR A=1 TO X
4040 SAVE
4050 PRINT
4060 NEXT A
4070 GOTO 2010
5000 L=0
5010 FOR J=1 TO I-1
5020 L=L+1
5030 PRINTB$(J)
5040 NEXT J
5050 IF L=32 THEN 1000
5060 IF L<32 THEN X=32-L
5070 X=X/3
5080 FOR A=1 TO X
5085 SAVE
5090 PRINT
5095 NEXT A
6000 GOTO 1000
```

ACTION, STRATEGY, AND FANTASY for the SERIOUS games player and his APPLE II

Brain Games - 1 demands ingenuity.

Two players bombard radioactive material with protons and electrons until it reaches critical mass and sets up a **Nucleer Reaction. Dodgam** *requires you to outmaneuver another player to get your pieces across the board first.* **Dueting Digits** *and* **Parrot** *challenges your ability to replicate number and letter sequences.* **Tones** *lets you make music with your Apple (16K) CS-4004 $7.95.* **Strategy Gemas** *end* **Brain Gamas** *are on one disk (16K) CS-4503 $14.95.*

THINK

Strategy Games - 1 keeps games players in suspense.

You and your opponent trail around the screen at a quickening pace attempting to trap each other in your **Blockade**. *A 7 category quiz game will certify you as a* **Ganius** *(or an errant knave!). Beginners will meet their master in* **Checkers. Skunk** *and* **UFO** *complete this classic collection (16K) CS-4003 $7.95*

Know Yourself through these valid self-tests.

Find out how your life style effects your **Life Expectancy** *or explore the effects of* **Alcohol** *on your behavior.* **Sex Rola** *helps you to examine your behavior and attitudes in light of society's concept of sex roles.* **Psychotharapy** *compares your feelings, actions, and phobias to the population's norms and* **Computer Literacy** *tests your microcomputer savvy. A fun and instructional package (16K) CS-4301 $7.95.* **Know Yoursalf** *and* **CAI Programs** *are on one disk (16K) CS-4503 for $14.95*

IMAGINE

You're in command in *Space Games* - 1.

Maneuver the TIE fighters into your blaster sights and zap them with your lasers to save the rebel base camp from annihilation in **Ster Wars. Rocket Pilot** *is an advanced real time take off and landing game. High resolution graphics, exploding saucers and sound effects add to the suspense as you repel the* **Saucer Invasion**. *Finally, a bonus graphics demonstration,* **Dynamic Bouncar** *(16K) CS-4001 $7.95.* **Speca Gemas** *and* **Sports Games** *are on one disk (16K) CS-4501 for $14.95*

ACTION

Sports Games - 1 puts you in the Apple World Series

Take the field in the **Graat American Computer Game**. *Mix up your pitches to keep the batter off balance. Move your fielders to snag the ball before he gets to first. Balls and strikes, double plays, force outs, and errors let you play with a realistic strategy. Also in the line up—* **Slalom,** *a championship downhill ski race,* **Torpedo Attey**, *and* **Darts** *(16K) CS-4002 $7.95.* **Speca Gemes** *and* **Sports Gamas** *are on one disk (16K) CS-4501 for $14.95*

It's easy to order *SENSATIONAL SOFTWARE* for your Apple II.

Send payment plus $1.00 shipping and handling in the U.S. ($2.00 foreign) to Creative Computing Software, P.O. Box 789-M, Morristown, N.J. 07960. N.J. residents add $1.00 sales tax. Visa, Master Charge and American Express orders may be called in toll free to 800-631-8112 (in N.J. 201-540-0445).

XREFER

XREFER stends for 'cross reference'. The BASIC progrem presented here premits the output of en essembler to be sorted end cross referenced. The cross reference listing cen be a very velueble tool when debugging mechine lenguege progrems.

Joel Swank
4555 SW 142nd # 188
Beaverton, OR 97005

When programming in assembly language the quality and features of the assembler being used can make a great deal of difference in how well the project proceeds. That's one reason professional programming departments are willing to spend a lot of money to buy and support large and powerful cross assemblers for their programming efforts. Most computer hobbyists though can only afford to use software that runs on their own machines. These assemblers for the most part, offer only the most basic features.

I bought MICRO-ADE(1) as an alternative to programming my MOS Technology KIM-1 microcomputer in machine language. MICRO-ADE is a large step up from machine language. It is also a large step down from the IBM-370 assembler to which I an accustomed. I soon found that most of the more advanced functions (expression evaluation, macros, relocatability, and conditional assembly) I could easily get along without. One thing I sorely missed though was the sorted cross-reference table. A cross-reference table is invaluable when debugging or modifying a program, especially when the program was written by someone else.

I implemented MICRO-ADE as my system assembler by modifying it to read source files from, and write object files to my disk system. It accepts unnumbered source files created by my system editor and generates its own line numbers. It creates object files that are loaded by a special load program. It has everything I need except a cross-reference table. To remedy this situation I wrote XREFER. XREFER is a program in MICROSOFT BASIC(2). It reads the same source files as MICRO-ADE, and produces a sorted label table giving the line number of the definition of each label and the line number of each reference to each label.

Implementation

The main task of a cross-reference program is data storage. It must be able to handle a variable number of labels, each with a variable number of references. The most obvious way to allocate the required storage in BASIC is to dimension 3 arrays, a one dimensional array for labels, a one dimensional array for definitions, and a two dimensional array for references. This will work, but one quickly runs into a problem. Most labels are referenced between 1 and 5 times in a assembly language program, but in most programs there will be one or more labels that are referenced 10, 20 or more times. To dimension the reference array large enough to hold the maximum number of references would use a large amount of memory. This would also waste a large amount of memory since most of the memory allocated for labels with fewer references would be unused. Allowing for a large number of references for each label also reduces the number of labels that can be handled in available memory. To dimension fewer than the maximum would result in an incomplete cross-reference. This problem can be overcome by dividing the reference storage into two 2 dimensional arrays. The first has an element for each entry in the label array and each element has room for a few references (5 or 6). The second is used as an array fo overflow arrays. It has only a few elements but each element has room for a lot of references (20 or 30). The reference array for a label is filled,the next available overflow array is chained to it, and all subsequent references to that label are stored in that overflow array. This allows more efficient use of available memory.

Larger source files will no doubt exceed available memory no matter how efficiently it is used. No matter how much memory is bought there will be a program that needs more. Is there no way to make the program handle an infinite amount of data? Yes there is! In this case the range of labels accepted into the table in any one reading of the file is limited. Then the file is read multiple times. Each reading will extract a different part of the entire table. For instance there may be only enough memory to store one-third of the labels in a

large assembly language program. XREFER can be run 3 times on the same source file. Labels beginning with A-G are cross-referenced in the first pass, H-S in the second, and T-Z in the third. The resulting 3 tables can then be joined to form a complete cross-referenced with enough passes of the file (Actually XREFER is limited to 32767 lines by the integer variables used to store the line numbers). Setting a range on the labels also allows operands other than than normal labels to be crossed-referenced. Immediate operands, absolute hex addresses, and data assignments can also be extracted if desired.

While the file is being read for the cross-reference table there is some other useful data that may be gathered. A table of opcodes and a count of their occurrences would also be interesting. For the MICRO-ADE defaults several addressing modes and optimizes others. Some address modes are implicit to the opcode. This means that those address modes cannot accurately be counted. Nevertheless the address modes that are counted correctly (IX, IY, IM) are worth the small space required.

XREFER is logically divided into four sections, initialization, data collection, sort and print, and subroutines. Listing 1 is the XREFER program listing. The initialization section prompts the user for options, allocates storage and opens the input file. The arrays for data storage are dimensioned according specifications input by the user. This allows the user to tailor memory usage to the source file being processed. When the cross-reference table option is selected XREFER prompts the user for the size of the label table, the number of references per label, the number of overflow arrays and the length of the overflow arrays. Determining the numbers to enter for each of the above is a matter of trial and error. The data collection subroutines have built-in overflow detection to aid the user. Mesages are printed when any of the arrays overflow. There are also overflow counters to record the number of times data is lost. These will give the user an idea of how much to increase the size of the arrays. When a program has too many labels for the available storage, the number of labels accepted can be limited by decreasing the range of labels accepted. It may take several runs to determine the correct parameters for a large program. XREFER also allows the user to select whether or not any of the three tables is built on a given run. If a table is not selected its storage is not allocated. The last thing requested is the filename. After the file is opened the assembler language source statements are read one at a time and the three parts (label, opcode and operand) are extracted. the comment field, if present, is ignored. The label is inserted into the label table and its definition line number saved. The opcode is stored and counted and the address mode extracted. The comment field, if present, is ignored. The label is inserted into the label table and its definition line number saved. The opcode is stored and counted and the address mode extracted and counted if present. The operand is then used to add a reference to the reference array (A new entry is made into the label table if necessary). The line numbers are generated as the lines are read in. When the end of the input file is reached, the sort and print section of XREFER is entered.

The Shell-Metzner sorting technique is used. Shell-Metzner requires a few more statements than the ubiquitous bubble sort but it executes in about a tenth of the time for a table of 200 labels. Any sorting algorithm requires switching of the data elements it is sorting. The labels in the label table in XREFER are connected logically to a data structure of definitions, references and overflow arrays. Switching labels would destroy this logical structure. Labels would end up with the wrong references. Moving the actual data around would require a lot of time and memory. Instead a special array (SRT%) of pointers is sorted. Before the sort, SRT% is initialized to 1, 2, 3,...etc. must be dimensioned at least as large as the number of elements being sorted. The sort comparison is made between elements indexed by SRT%. Then the pointers in SRT% are switched if necessary instead of the actual data. After sorting is finished, the SRT% array is used to index the data for printing. The labels are printed in alphabetical order with their definitions and references. The same technique is used for both the opcode table and the label table. The address mode table is a static table and is not sorted. Ater all the requested tables are printed XREFER gives the user the opportunity to repeat the printing section to get another copy of the tables. XREFER can also be restarted at line 7200 to print the tables.

Operation

Listing 2 is a sample run of XREFER. XREFER prompts the user for each parameter. In this run the arrays were purposely dimensioned too small to show the error messages generated when they over flow and what to change to correct problem. Note that answering 'N' to the ' SYMBOL TABLE (Y or N)?' will cause all other questions about the symbol table to be ommited. Also answering 0 to 'NUMBER OF OVERFLOW ARRAYS?' will effectively delete overflow processing from the program.

I use XREFER to document all of my larger assembly language programs. I use the cross-reference often during testing. With it I can quickly locate every reference to a data area and every place a subroutine is called. XREFER takes longer to generate the cross-reference listing than MICRO-ADE takes to assemble the same source file but the resulting cross-reference table is well worth the time.

(1) MICRO-ADE is an assembler for the 6502 microprocessor. It is sold by MICROWARE Ltd. 27 Firstbrook Rd. Toronto, Ontario, Canada M4E 2L2. It does not use the standard MOS Technology syntax.

(2) XREFER is written in 9 digit MICROSOFT BASIC. It is distributed by MICRO-Z company Box 2426 Rolling Hills California 90724. It has been extended to add disk I/O capability.

LISTING 1: XREFER implemented in MICROSOFT BASIC for the 6502 microprocesser. XREFER uses standard BASIC syntax except for the disk I/O related commands. The disk I/O commands are implemented as standard BASIC commands with the postfix character #. DIM # allocates a buffer for the file. GET

opens the file for input. END # sets the statement number to be executed when the end of the file is reached. INPUT # reads line of the file. Variables followed by a # are integer variables. Integer variables are used wherever possible to save storage. Integer arrays require only 2 bytes of storage per entry while floating point arrays require 5 bytes per entry.

Listing 2: A run of XREFER genterating all three tables for a small assembly language program.

```
I 151

1000 REM      XREFER
1100 REM
1200 REM
1300 REM INITIALIZATION
1400 REM
1500 CIM# 1
1600 TRUE=-1:FALSE=0
1700 PRINT "ENTER OPTIONS"
1800 INPUT "SYMBOL TABLE(Y OR N)";ANS$
1900 IF ANS$<>"Y" THEN 3000
2000 INPUT "NUMBER OF SYMBOLS TO DIMENSION";NUM
2100 INPUT "NUMBER OF REFERENCES TO DIMENSION";XR
2200 INPUT "ENTER NUMBER OF OVERFLOW ARRAYS";OV
2300 IF OV<=0 THEN 2600
2400 INPUT "ENTER LENGTH OF OVERFLOW ARRAYS";OL
2500 DIM ROVFL%(OV,OL)
2600 INPUT "SYMBOL RANGE LO";RL$
2700 INPUT "SYMBOL RANGE HI";RH$
2800 LT%=TRUE
2900 DIM LABEL$(NUM), DEF%(NUM), REF%(NUM,XR)
3000 INPUT "OPCODE TABLE(Y OR N)";ANS$
3100 IF ANS$ <> "Y" THEN 3400
3200 DIM CODE$(60), CCNT%(60)
3300 CT%=TRUE
3400 INPUT "ADDRESS MODE TABLE(Y OR N)";ANS$
3500 IF ANS$ <> "Y" THEN 4000
3600 DIM MODE$(9), MCNT%(9)
3700 MT%=TRUE
3800 MODE$(7)="Z":MODE$(8)="A":MODE$(9)="NONE"
3900 FORI=0 TO 6:MODE$(I)=MID$("AYAXZXZYIMIXIY",I*2+1,2):NEXT
4000 INPUT "ENTER FILENAME";F$
4100 IF NOT CT% AND NOT LT% THEN 4500
4200 A=60
4300 IF NUM>A THEN A=NUM
4400 DIM SRT%(A)
4500 LINNO%=1
4600 SET# 1 FI$:END# 1 GOTO 7200
4700 REM
4800 REM          DATA COLLECTION
4900 REM
5000 INPUT# 1 LINE$
5100 IF LINE$="" OR LEFT$(LINE$,3)="   " THEN 6700
5200 REM GET LABEL, OPCODE AND OPERAND
5300 GOSUB 18100
5400 REM STORE LABEL
5500 IF NOT LT% THEN 5900
5600 T$=LEFT$(LABEL$,1):IF T$>=RL$ AND T$<=RH$ THEN GOSUB 17000
5700 REM STORE REFERENCE
5800 T$=MID$(OP$,1,1):IF T$>=RL$ AND T$<=RH$ THEN GOSUB 20500
5900 IF CODE$="" THEN 6700
6000 MODE$=MID$(CODE$,4,2)
6100 IF NOT CT% THEN 6400
6200 REM COUNT OPCODE
6300 GOSUB 23200
6400 IF NOT MT% THEN 6700
6500 REM COUNT MODE
6600 GOSUB 24000
6700 LINNO%=LINNO%+1
6800 GOTO 5000
6900 REM
7000 REM          SORT AND PRINT DATA
7100 REM
7200 IF NOT LT% THEN 11700
7300 GOSUB 24800
7400 REM
7500 REM SORT XREF TABLE
7600 REM
7700 FOR I=1 TO NUM:IF LABEL$(I)<>"" THEN NEXT
7800 N%=INT(I-1):M%=N%
7900 M%=M%/2
8000 IF M%=0 THEN 9500
8100 K%=N%-M%
8200 J%=1
8300 I%=J%
8400 L%=I%+M%
8500 IF LABEL$(SRT%(I%))<=LABEL$(SRT%(L%)) THEN 8900
8600 TMP%=SRT%(L%):SRT%(L%)=SRT%(I%):SRT%(I%)=TMP%
8700 I%=I%-M%
8800 IF I%>=1 THEN 8400
8900 J%=J%+1
9000 IF J%>K% THEN 7900
9100 GOTO 8300
9200 REM
9300 REM PRINT XREF TABLE
9400 REM
9500 I=1
9600 PRINT:PRINT "SYMBOL","DEFINED","REFERENCES"
9700 S=SRT%(I):IF LABEL$(S) = "" THEN 11700
9800 PRINT LABEL$(S),DEF%(S),
9900 FOR J=0 TO XR
10000 IF REF%(S,J)=0 THEN 11200
10100 IF POS(X)>65 THEN PRINT:PRINT"","",
10200 IF REF%(S,J)>0 THEN 11000
10300 L=-REF%(S,J)-1
10400 FOR K=0 TO OL
10500 IF ROVFL%(L,K)=0 THEN 11200
10600 IF POS(X)>65 THEN PRINT:PRINT"","",
10700 PRINT ROVFL%(L,K);
10800 NEXT
10900 GOTO 11200
11000 PRINT REF%(S,J);
11100 NEXT J
11200 PRINT
11300 I=I+1:IF I<=NUM THEN 9700
11400 REM
11500 REM SORT OPCODE TABLE
11600 REM
11700 IF NOT CT% THEN 15100
11800 GOSUB 24800
11900 FOR I=1 TO 60:IF CODE$(I)<>"" THEN NEXT
12000 N%=INT(I-1):M%=N%
12100 M%=M%/2
12200 IF M%=0 THEN 13700
12300 K%=N%-M%
12400 J%=1
12500 I%=J%
12600 L%=I%+M%
12700 IF CODE$(SRT%(I%))<=CODE$(SRT%(L%)) THEN 13100
12800 TMP%=SRT%(L%):SRT%(L%)=SRT%(I%):SRT%(I%)=TMP%
12900 I%=I%-M%
13000 IF I%>=1 THEN 12600
13100 J%=J%+1
13200 IF J%>K% THEN 12100
13300 GOTO 12500
13400 REM
13500 REM   PRINT OPCODE TABLE
13600 REM
13700 PRINT:PRINT "OPCODE USAGE TABLE"
13800 J=1:FOR I=1 TO 60
13900 FOR K=1 TO 4
14000 T$=CODE$(SRT%(J))
14100 IF T$="" THEN 15100
14200 T=LEN(T$):IF T<3 THEN T$=T$+" ":GOTO 14200
14300 PRINT "     ";T$;CCNT%(SRT%(J)),
14400 J=J+1
14500 NEXT K
14600 PRINT
14700 NEXT I
14800 REM
14900 REM   PRINT ADDRESS MODE TABLE
15000 REM
15100 IF NOT MT% THEN 16200
15200 PRINT:PRINT:PRINT "ADDRESS MODE OCCURENCES"
15300 MODE$(7)="Z ":MODE$(8)="A "
15400 J=0
15500 FOR K=1 TO 3
15600 IF J>9 THEN 16200
15700 PRINT "     ";MODE$(J);MCNT%(J),
15800 J=J+1
15900 NEXT
16000 PRINT
16100 GOTO 15500
16200 PRINT:PRINT:PRINT:INPUT "REPEAT";ANS$
16300 IF ANS$="Y" THEN 7200
16400 END
16500 REM
16600 REM SUBROUTINES START HERE * * *
16700 REM
16800 REM STORE LABEL
16900 REM
17000 FOR I=1 TO NUM
17100 IF LABEL$(I)=LABEL$ THEN 17600
17200 IF LABEL$(I)="" THEN 17500
17300 NEXT
17400 O1=O1+1:PRINT"TOO MANY LABELS":RETURN
17500 LABEL$(I)=LABEL$
17600 DEF%(I)=LINNO%
17700 RETURN
17800 REM
17900 REM PARSE FOR LABEL OPCODE AND OPERAND
18000 REM
18100 CODE$="":OP$=""
```

```
18200 LABEL$="":IE LEFT$(LTNE$,1) <> " " THEN 18500
18300 K=2
18400 COTO 19100
18500 FOR K=1 TO 6
18600 IF M1O$(LTNE$,K,1)= " " THEN 19000
18700 LABEL$=LABEL$+MTO$(LTNE$,K,1)
18800 NEX1
18900 RETURN
19000 K=K+1
19100 EOR J=K 10 K+5
19200 T$=MIO$(LTNE$,J,1):1F T$="" THEN 20100
19300 IE T$=" " 1HEN 19600
19400 COOE$=COOE$+T$
19500 NEXT
19600 J=J+1
19700 EOR K=J TO J+6
19800 T$=MTO$(L1NE$,K,1):TF T$="" OR I$=" " THEN 20100
19900 OP$=OP$+T$
20000 NEXT
20100 RETURN
20200 REM
20300 REM COUNT REEERENCE
20400 REM
20500 EOR 1=1 TO NUM
20600 IE LABEL$(I)=OP$ THEN 21100
20700 1E LABEL$(1)="" THEN 21000
20800 NEXT
20900 01=01+1:PRINT"TOO MANY LABELS":RE1URN
21000 LABEL$(I)=OP$
21100 EOR J=0 TO XR
21200 IE REE%(1,J)=0 1HEN REE%(1,J)=LINNO%:RE1URN
21300 NEXT
21400 1E OV<1 1HEN PR1NT"NO OVERELOW ARRAYS": 4=C4+1:RETURN
21500 J=J-1
21600 1E REF%(I,J)<0 1HEN 22400
21700 FOR K=0 TO OV:IE ROVEL%(K,0)=0 1HEN 22900
21800 NEXT:02=02+1:PRINT"NOT ENOUGH CVERFLOW ARRAYS":RETRRN
21900 REM SET UP CHA1N
22000 ROVFL%(K,0)=REE%(1,J)
22100 REE%(I,J)=-K-1
22200 ROVEL%(K,1)=LINNO%:RE1URN
22300 REM ADD 10 OVERELOW
22400 K=-REF%(I,J)-1
22500 EOR L=1 TO OL:IE ROVFL%(K,L)=0 1HEN 22700
22600 NEXT:03=03+1:PRIN1"CVERELOW ARRAYS NOT LONG ENCHGH":RETURN
22700 ROVEL%(K,L)=L1NNO%:RETURN
22800 RE1URN
22900 REM
23000 REM STORE AND COUNT OPCODE
23100 REM
23200 CODE$=LEFT$(COOE$,3):FOR 1=1 TO 59
23300 IE COOE$(T)=COOE$ GOTC 23700
23400 1F COOE$(I)="" THEN 23600
23500 NEXT
23600 CODE$(1)=CODE$
23700 CCN1%(I)=CCNT%(T)+1
23800 RETURN
23900 REM
24000 REM    COUNT ADDRESS MODE
24100 REM
24200 FOR I=0 TC 8
24300 IE MOOE$(I)=MOOE$ 1HEN 24500
24400 NEXT
24500 MCNT%(1)=MCNT%(I)+1
24600 RETURN
24700 REM INTT SORT POINTER MATR1X
24800 EOR I=1 10 A:SRT%(1)=I:NEXT:RETURN
JK

RUN
FNTER OPTIONS
SYMBOL TABLE(Y OR N)? Y
NUMBER OF SYMBOLS TO DIMFNSION? 100
NUMBER OF RFFERENCES TO DIMENSION? 4
ENTER NUMOER OF OVERFLOW ARRAYS? 4
FNTER LENCTH OF OVERFLOW ARRAYS? 25
SYMBOL RANGE LO? A
SYMBOL RANGE HI? Z
OPCODE TABLE(Y OR N)? Y
ADDRESS MODE TABLE(Y OR N)? Y
ENTER FILENAMF? 2/LOADQ
TOO MANY LABELS
TOO MANY LABELS
TOO MANY LABELS
TOO MANY LABELS
TOO MANY LABELS
TOO MANY LABELS
H
BREAK IN  17100
OK
RUN
ENTER OPTIONS
SYMBOL TABLE(Y OR N)? Y
NUMBER OF SYMBOLS TO DIMENSION? 110
NUMBER OF REFERENCES TO DIMENSION? 4
ENTER NUMBER OF OVFRFLOW ARRAYS? 4
ENTER LENGTH OF OVERFLOW ARRAYS? 25
SYMBOL RANGF LO? A
SYMBOL RANGE HI? Z
OPCODE TABLE(Y OR N)? Y
ADDRESS MODE TABLE(Y OR N)? Y
ENTER FILENAMF? 2/LOADQ
OVERFLOW ARRAYS NOT LONG ENOUGH
OVERFLOW ARRAYS NOT LONG ENOUGH
OVERFLOW ARRAYS NOT LONG ENOUGH
@
BREAK IN  8800
OK
RUN
ENTER OPTIONS
SYMBOL TABLE(Y OR N)? Y
NUMBER OF SYMBOLS TO DIMENSION? 110
NUMBER OF REFERENCES TO DIMENSION? 4
ENTER NUMBER OF OVERFLOW ARRAYS? 4
FNTER LENGTH OF OVERFLOW ARRAYS? 35
SYMBOL RANGE LO? A
SYMBOL RANGE HI? Z
OPCODE TABLE(Y OR N)? Y
ADDRESS MODE TABLE(Y OR N)? Y
ENTER FILENAMF? 2/LOADQ

SYMBOL  DEFINED  REFERENCES
ADTMPH    32       125
ADTMPL    31       123
ALLOCX    92
ALTH      30       293
ALTL      29       290
ARMBUF    381
BACKX     80       336
BADADD    218      203  212
BADFIL    177      107  130  134  150
BADGET    276
BADRET    180      128  143  168
BCSBAD    149      113
BINDEX    86       246  259  296  334
BUFADD    82       288
BUFF      422      114  116
BUFFER    54
BUFPTH    41       386
BUFPTL    40       383
BUFPTR    39       255
BYTBYE    216      199  210
CHRSAV    43       208  213
CRLF      67       141
CTKP      26
CURCHR    34       326
```

```
DIRPTR   33   343  346  350  353  360  363
DPTH     28
DPTL     27
DRIVE    78   285  328
DRVSAV   35   330
DSDR     15   287
DSEC     17   284  317
DSTK     16   281  314
ENDOFF   98   135
ENDOFI   94
EOF      215  201
ERRET    219  225  227
FCBPTH   38   111
FCBPTL   37   109
FCOPTR   36   244  247  250  260  270  273  280  283
              286  289  292  298  301  304  309  312
              315  318  329  335  337  339  341  345
              348  355  358  365  368  371  382  385
              388
FILERR   95
FILTYP   83   299
FIRST    142  133
FLSC     19   311
FLTK     18   308
FNAME    77
FTYP     23   300
FWDC     60
GETADD   63   121
GETBYT   198  142  148
GETCHR   242  127  209
GETDRV   64   327
GETEOF   275  245
GETRET   277  295
GETSEC   268
INOUT    87
INVADD   93
INVCMD   91
INVOFF   100
LEN      20   303
LENGTH   81   249  302
LIMITX   85   364
LOADER   103
LODLUP   167  173  175
LRET     145  122
MESGS    394  185
MOVEFB   307
NEXTX    79   269  279
NOAD     232  229
NOCHR    263  253
NOEND    147  144
NOEOF    279  271  274
NXTCHR   254  251
NXTOKN   65   105  120
OBJFCB   421  110  115  117  119  131  422
OBJPTH   48   137  147  156  159  161  174
OBJPTL   47   139  151  153  155  163  172
OBJPTR   46   171
OFFSEH   50   126  157
```

```
OFFSEL          49      124   154
OKYDOK          1Ø8     1Ø6
OPEN            326     112
OPNLUP          37Ø     373
OPNRET          375     332
ORGOFF          99      159
OUTCH           7Ø      188
OUTSP           68      165
PACKX           224     2Ø2   211
PDUN            191     187
PMSG            185     136   16Ø   178   19Ø
PRTBYT          69      138   14Ø   162   164
PSTH            25
PSTL            24
READSC          252     248
REORG           135     169
RSEX            58      294
SAVHEX          45
SAVOBJ          44
SEARCH          61      331
STARTX          84      354
TGTH            22
TGTL            21
THEEND          418
WSEX            59
WTDIR           62
XSAVE           42
ZERO            97      17Ø   297   333

OPCODE USAGE TABLE
     *    31          =    24        ADC  3       AND  1
     ASL  4           BCS  12        BEQ  7       BMI  2
     BNE  1Ø          BPL  2         CLC  8       CMP  11
     DEY  4           INC  2         INX  1       INY  14
     JMP  1           JSR  23        LDA  38      LDX  4
     LDY  2Ø          ORA  1         PHA  3       PLA  3
     RTS  7           SEC  1         STA  41      TAY  2
     TYA  1

ADDRESS MODE OCCURENCES
     AY  Ø            AX  1          ZX  Ø
     ZY  Ø            IM  38         IX  Ø
     IY  41           Z   Ø          A   4
     NONE  197

REPEAT? N

OK
```

800 ATARI®

$819

PET

16K $895
32K $1195

400 ATARI®

$499

APPLE*

MICROSOFT ADVENTURE
The original of ADVENTURE written for the DEC-10 systems is now available for the APPLE. Explore Colossal Cave for treasures while avoiding the dangers hidden within its many passages. 130 different rooms. 15 treasures, and characters ranging from helpful to deadly await you within the cave. Be careful where you step, and also who's behind you!
32K disk machine language $29.95

ANDROID NIM
by Leo Christopherson
The game that made Leo Christopherson famous is now available for the APPLE! The improved graphics and color of the APPLE make the game even better. Try to be the last one to shoot the androids on the screen. If you do, you win! Also includes realistic sound effects.
24K cassette machine language .. $14.95

MAGIC PAINT BRUSH
Hi-Res graphics package plus! Draw Hi-Res pictures using all APPLE's colors. Connect any points on screen, fill areas, plot, rotate, and scale shape, or 'paint' with a set of 9 brushes. Also comes with Shape Table Designer and 2 demo programs. Slot Machine and Applesott Invaders.
32K disk Applesoft-ROM $29.95

THREE-D
You don't have to be an engineer or scientist to have high resolution graphics for your computer! This program permits rotation, scaling, shift, distortion, and combination of three dimensional graphics on the screen. MP Software.
48K disk Applesoft-ROM $29.95

ATARI*

STAR RAIDERS
The best! A ROM cartridge holds the game. A fast-paced full-color, space battle in which you must defeat the enemy Zylon ships while protecting your home bases. Real-time action and effects make this game the best space game available. Sixty levels of rating from Garbage Scow 4th Class to Commander make for continuously exciting play.
ROM cartridge $59.95

3D GRAPHICS
by Tim Hayes
High quality graphics program for the ATARI computer allows you to rotate, distort, shrink, and combine three dimension graphic projections on the screen. With the high resolution abilities of the ATARI, one of the finest graphics packages available anywhere!
16K cassette.................... $29.95

WALL STREET CHALLENGE 6402
A computer simulation of the Stock Exchange is easy to play and always challenging. Invest in stocks, and try to make it big!
8K and 16K version on one cassette
.... $19.95

ALL STAR BASEBALL 6401
Two players face each other, one at batand the other pitcher and outfield. Innings, balls, strikes, and a variety of plays make for an exciting game. Joysticks are optional.
8K and 16K versions on one cassette.
.... $19.95

PET*

STARFLEET ORION
Command a starfleet! 2 player game system includes rule book, battle manual, control sheets. Two programs. 22 space ship types and 12 play tested scenarios.
8K cassette $19.95

RESCUE AT RIGEL
Search the moon base and rescue Delilah Rookh from the High Tollah. Automatic Simulations.
24K cassette.................... $19.95

MORLOC'S TOWER
Match wits with the evil wizard and try to defeat him! Automated Simulations.
32K cassette.................... $14.95

TIME TREK
by Brad Templeton from Personal Software. Real time action. Star Trek type game with sound effects. There are no 'turns'. The action continues whether you move or not. You and the Klingons can move, steer, and fire at the same time.
8K cassette $14.95

This is only a very small sample of our product line. For a complete selection, send $1 for our catalog of hardware, software, and publications and receive a $2 credit toward your first order.

APPLE, ATARI, and PET are trademarks of Apple Computer Co., Warner Communications, and Commodore, respectively

The Software Exchange 6 SOUTH ST., MILFORD, NH 03055

To order: Call Toll-Free 1-800-258-1790 (In NH call 673-5144)

master charge VISA®

The Software Exchange & Hardside (Div. of Robitaille & Sons, Enterprises, Inc.), SoftSide Publications

In this issue of MICRO, the Ohio Scientific Small Systems Journal presents a system overview of OS-65U Level I and a very informative article on expanding OS-65D mini-floppy BASIC.

OS-65U Level I allows the setup of a simple, cost-effective, multi-terminal network using a single disk based computer in concert with several personal computers. The system is extremely well suited for the educational environment and demonstrates some of the 'hidden power' of the personal computer.

The article on expanding mini-floppy BASIC demonstrates a clever method which allows up to 26 new reserved words to be added to BASIC.

As always, reader comments on article content are always welcome. Please submit suggestions, or any other contributions, to:

Ohio Scientific, Inc.
Small Systems Journal
1333 South Chillicothe Rd.
Aurora, Ohio 44202

OS 65U LEVEL I — UPLOADING AND DOWNLOADING ON A MULTI — TERMINAL SYSTEM

Even small systems can take advantage of the storage capabilities of any one of a group of Challenger computers (C-1P's, C-4P's, and/or C-8P's). This feature permits networking of computers, sharing a central file system, and even information interchange between terminals.

The OS-65U operating system can service and support several satellite "personal computers" from a central host computer. Each satellite computer can be a C-1P, a C-4P, or a C-8P, and for the remainder of the article will also be referred to as a "terminal". The terminal can stand alone with no reduction in its capabilities or it can use the resources of the host computer to extend its capabilities. Hardware modifications, readily performed by your OSI dealer, will be required. The satellite computers/terminals, when initially ordered can be specified with an "Option-11" for the C-XP systems. The Option-11 allows up/down load as well as retaining normal cassette I/O. The host system requires installation of a multiple I/O port board, designated as CA-10L8 for 8 ports. In general, any disk based system can serve as the host computer. It is convenient to choose the one with the greatest disk storage capability, In order to present the maximum increase in storage to each terminal.

Each satellite computer, whether the C-1P, C-4P or C-8P, retains all the features of the stand-alone computer. These features include 8K MICROSOFT (R) BASIC in ROM, the ability to SAVE and LOAD cassette programs, and access to all the computer's memory and accessories. For example, the home control features of a C-4P MF could be enjoyed while using the computer for computer aided instruction.

Programs and data files can be downloaded from or uploaded to the host computer in a Level I Multi-Terminal System. This feature permits applications such as Computer Based Education, with the ability to access the lesson or course on the host computer while retaining the powerful BASIC programming capability at each computer terminal. Sharing data and exchanging programs while retaining isolation of each independent, giving its user the full resources of the computer at that station. The CPU (Central Processor Unit) of each station is totally available for the user, since it does not have to timeshare its resources with the host. The benefits of fast response, high data transfer rates, and low cost are not compromised.

USE:

To each terminal on the Multi-Terminal System, the host computer will function as a high speed serial port which can be addressed by a filename. Each terminal uses its serial port at a clock rate set by jumpers in the host computer (with data rates up to 19.2 Kilobaud!)

To use the Multi-Terminal System, BOOT up the host system and RUN the program MULTI. All current OS-65U Systems contain this program on disk file. Now, BOOT up your terminals(s), with the Cassette/Level 1 switch positioned at LEVEL 1. If you do nothing else, your terminal is a stand alone computer. Let's take advantage of this status to enter a very short program.

```
NEW
10 PRINT "TEST MESSAGE":END
```

When you type

```
SAVE
```

the facilities of the host computer will be made available assuming the terminal switch in the LEVEL I position. Since we wish to save this program we type

```
REM S FILNAM
```

where FILNAM is the file name of an available disk file on the host computer. The host computer expects the next entry to be

```
LIST
```

which effects the transfer of the program to the host computer's file, FILNAM. To discontinue transfer capability to the host computer, type, for example, the entry

```
LOAD <CARRIAGE RETURN>
```

then

```
<SPACE>
```

The symbol <SPACE> denotes a blank space. Similarly, the symbol <CARRIAGE RETURN> denotes a carriage return. These symbols will be used when there is some chance of ambiguity of notation. Otherwise, <CARRIAGE RETURN> is assumed to terminate a keyboard entry. If we now enter

```
NEW
```

we will clear the workspace on our terminal. We can check this by typing

```
LIST
```

To download our program from the host computer's disk file, we again get the services of the host computer by entering the command

```
SAVE
```

and then

```
REM L FILNAM
```

then

```
LOAD
```

The file, FILNAM, will be transferred from the host computer's disk and displayed during transfer to the terminal, as we can observe by typing

```
<SPACE>LIST
```

The short program should be listed on our terminal screen. If we had wished to list on the host computer's printer, the command would have been

SAVE
REM P
LIST
LOAD
<SPACE>
Note: LOAD, <CARRIAGE RETURN>, <SPACE> terminate link.

Provisions are made in the program MULTI to disconnect a terminal which has requested services of the host computer (by typing SAVE) but has not finished its request by entering
REM L FILNAM
After approximately 13 seconds of inactivity, the program MULTI will assume that no further activity is expected from the calling terminal, and the host computer will again scan the terminals for input.

APPLICATIONS:

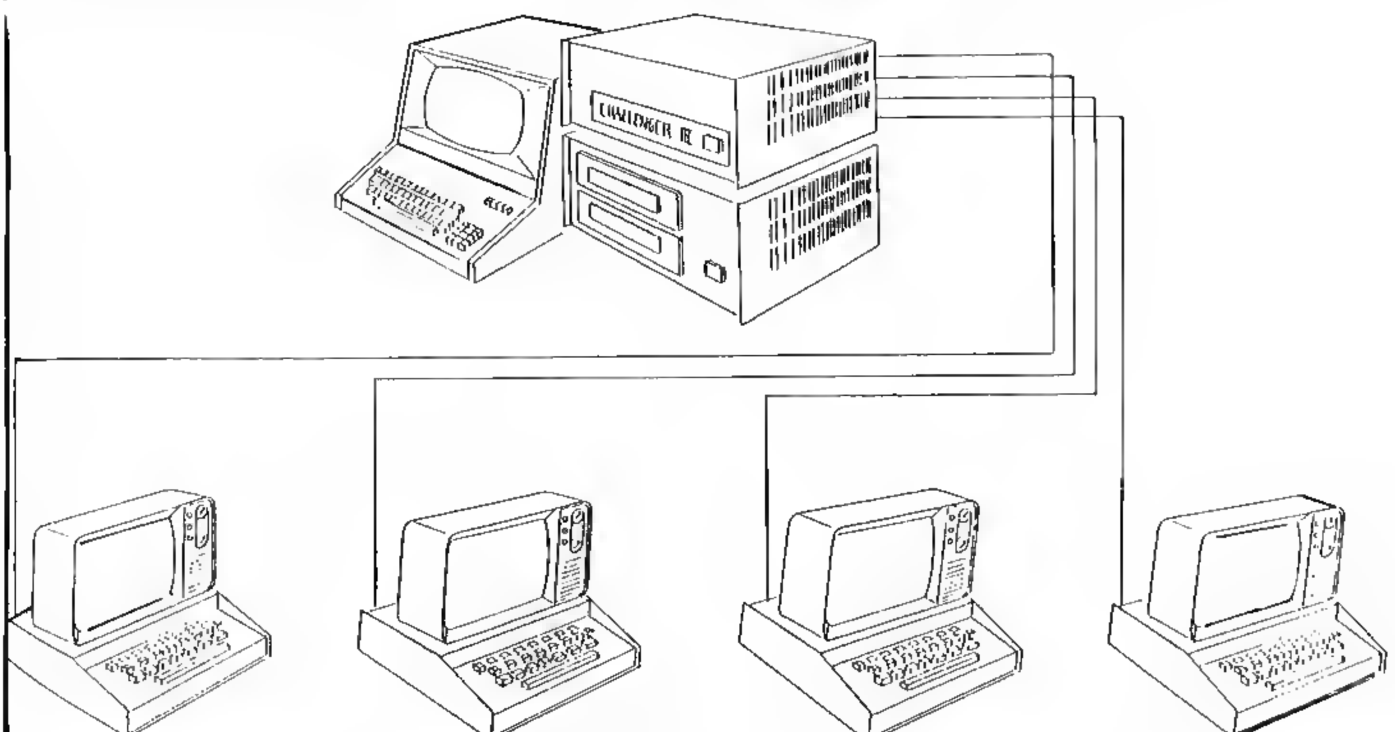
Storing copies of programs, such as educational materials, and uploading and downloading the programs to each terminal makes these programs available within an educational network. (These same benefits of uploading and downloading could prove equally useful in a small business environment.)

In a typical application in education, a C-8F DF might be used as a host computer while four satellite computers, say C-1P's, serve as individual student stations. Although each satellite costs about half the cost of a dumb terminal, it possesses more abilities than many nominally intelligent terminals. The program MULTI would permit each terminal to request downloading of the current lesson. For simplicity, the instructor may wish to modify MULTI to permit automatic downloading to ease lesson startup for less experienced students. Each student could then save his/her lesson onto cassette for future use after the lesson is completed.

In a typical lesson, the student may have need of the computing power of BASIC. A null response to a lesson question will return the student to BASIC. After completing his/her calculations in BASIC the student could return to the educational program with the results of the calculation in hand.

In a similar manner, student responses can be automatically stored for instructor review by writing the educational program with storage of the answers in an answer array. By SAVEing the student's program in a student file on the host computer, individual answers can be reviewed and student progress assessed by the instructor. These possible features are all within the flexibility of BASIC programming. These features allow the power of a sophisticated computer aided instruction system to by built on the resources of your OS 65U host system and the simple BASIC programming of MULTI.

Since the satellite stations can be run with high baud rates and downloaded programs can themselves request downloading, then it is possible to use the extensive graphics support which is available from OSI within educational program with the results of the calculation in hand.



TYPICAL LEVEL 1 SETUP

These ideas are intended to point out the power of what appears to be a relatively simple program.

Summary

Several methods are available to obtain source materials, including cassette or keyboard entry, and disk entry from the terminal or host computer. the methods provide a variety of ways to transfer materials between systems.

Since MULTI is written in BASIC, it provides the flexibility to support your specialized Level I needs, providing specialized uploading and downloading, logging, and branching to special servicing programs by the methods suggested.

Flexibility characterizes the LEVEL I System. It provides the benefit of sharing common programs and using central files while retaining the dedicated computer use at each terminal. It provides a case of "having your cake and eating it too!"

Adding New Reserved Words to BASIC

In this article, we describe a method that can be used by the assembly language programmer to add new reserved words to Ohio Scientific Microsoft BASIC on mini-floppy disk based computers using OS-65D B3.x. This method involves mimimal changes to the BASIC interpreter. These changes can be accomplished by seven POKEs. Using this method, one can add up to 26 new, words, each of which is a single letter followed by an asterisk (*). To simplify the assembly language code, we require that each new reserved word contain no embedded blanks. Each new word can be executed either in the immediate mode or from a running program.

Ohio Scientitic Microsoft BASIC is implemented by an interpreter. This means that the BASIC program is stored in memory in ASCII, just as it was entered from the keyboard (with exceptions which we will describe later). When a BASIC program is running, the interpreter (a machine language program) examines each line, executes the appropriate code and then advances to the next line. To do this, the interpreter maintains a pointer, which we will call TXTPTR, that points to the area in memory which contains the BASIC statement currently being interpreted by the Interpreter. Note: A pointer is a word in memory that contains an address.

TXTPTR is a 16-bit word at $C7 and $C8 on page 0. The low byte of the address is at $C7 and the highbyte is at $C8. Note: $ is 'shorthand' meaning hexadecimal. When the interpreter begins to scan a line of BASIC code, TXTPTR points to the first non-blank character on the line. After interpreting the line, TXTPTR must be incremented until it points to the byte containing the terminator for the line, either $00 or $3A (carriage return or colon, respectively).

As we said previously, a BASIC program is stored in ASCII as it was entered from the keyboard. One exception is that a carriage return is stored as $00. The other exception is that all reserved words (PRINT, NEXT, GOSUB, etc.) and all operators (*, + , AND, SIN, etc.) are "tokenized", that is stored in one byte in a special code which is not standard ASCII. The token for an asterisk is $A5.

When a line of a BASIC program is stored in memory, the first byte after the line number is one of the tokens. The one exception to this is a LET statement which omits the reserved word LET. That is, LET X=0 or X=0. Hence, if the first group of characters on a line is a word which is not a BASIC reserved word, the interpreter branches to the code for LET. This is where we inserf a JSR to our new code which will look for new words and execute appropriate code if one is found.

The code for LET begins at decimal address 2470 (OS-65D V3.x on mini-floppy). The first three lines of this code are, in machine language, $20, $2E, $0F, $85, $96, $84, $97. We replace these seven bytes with hex 20, 00, 50, EA, EA, EA, EA. This calls a machine language subroutine at $5000.

Address $5000 is where we will put out new code. this address can be changed to any other available address by the user. The changes in these eight bytes can be accomplished by:

```
POKE  2470
POKE  2471.0
POKE  2472, 80
POKE  2473, 234
POKE  2473, 234
POKE  2475, 234
POKE  2476, 234
```

An appropriate place for these POKEs is in BEXEC*. They can also be put at the beginning of a BASIC program which contains a new reserved word. In our sample programs we put our new code at $5000. After the new code is assembled, it can be stored on disk with a DISK!"SA---" instruction and then broughf into memory by a DISK!CA--". Thus, on the disk which contained our sample programs, we added to BEXEC* the seven POKEs above and a DISK!"CA--" instruction.

In Listing 1 we introduce one new word, C*, which initiates a machine language screen clear. The program is, in outline, the tollowing:

Step 1) Check the second character on the line to see if it is the token for an asterisk. Then see if the first character is a C. If either of these fail, branch to BACK where we executed the machine code that was deleted from LET, then RTS back to LET.

Step II) If the line is C* then execute the screen clear code.

Step III) Add 2 to TXPTR.

Step IV) At this point we want to return to the point from which LET was called, so we can proceed to the next line. Execution of an RTS, however, will take us back to LET and a syntax error will result. Thus, we first execute PLA:PLA to remove one address from the stack and then RTS.

In the next example we insert two reserved words: C* as above and S* which will act as a switch to enable or disable the scrolling of

the screen after a PRINT. The effect of S* is the same as:

X = PEEK(9770)
IF X = 64 THEN POKE 9770,0
IF X = 0 THEN POKE 9770, 64

Listing 1

```
10  00C7=           TXTPTR=$C7
20  00A5=           TOKEN=$A5
30                ;
40  5000            *=$5000
50                ;
60  5000 A001     NEWDRO  LDY #1
70  5002 B1C7             LDA (TXTPTR),Y  get 2nd chr of the line
80  5004 C9A5             CMP #TOKEN
90  5006 D02E             BNE BACK        if not ster token then RTS
100 5008 88               DEY
110 5009 B1C7             LDA (TXTPTR),Y  got 1st chr of the line
120 500B C943             CMP #'C
130 500D D027             BNE BACK        if not "C" then back to LET
140               ;
150               ;
160               ;code for new reserved word begins here
170               ;
180 500F A920     NEWCDE  LDA #32         ASCII for e blank
190 5011 A000             LDY #0          ready for indexed STA
200 5013 A208             LDX #8          no. of pages on screen
210 5015 990000 PUTIT     STA $D000,Y
220 5018 C8               INY
230 5019 D0FA             BNE PUTIT
240 501B EE1750           INC PUTIT+2     if Y rolls over then chenge
250 501E CA               DEX '           pege
260 501F D0F4             BNE PUTIT
270 5021 A9D0             LDA #$D0        restore for
280 5023 8D1750           STA PUTIT+2     next call
290               ;
300               ;
310               ;need to update TXTPTR before return to BASIC
320               ;
330 5026 A5C7     UPDATE  LDA TXTPTR
340 5028 18               CLC
350 5029 6902             ADC #2
360 502B 85C7             STA TXTPTR
370 502D A5C8             LDA TXTPTR+1
380 502F 6900             ADC #0            edd the carry if it's there
390 5031 85C8             STA TXTPTR+1
400               ;now TXTPTR points to the end-of-line marker
410               ;
420               ;there are two return eddresses on the stack
430               ;pull off the top one so thet we return to the
440               ;place where LET wes called, instead of to LET
450               ;
460 5033 68               PLA
470 5034 68               PLA
480 5035 60               RTS
490               ;
500               ;
510               ;BACK is the machine code that was deleted from LET
520               ;and repleced by JSR $5000
530               ;
540 5036 20       BACK    .BYTE $20,$2E,$0F,$85,$96,$84,$97
540 5037 2E
540 5038 0F
540 5039 85
540 503A 96
540 503B 84
540 503C 97
550 503D 60               RTS
560                       .END
```

Listing 2 is outlined as follows:

Step I) Compare the second character on the line and the token for an asterisk. If it isn't, then branch to BACK as in the first program.

Step II) If it is an asterisk, then enter a loop which compares the first character on the line and the entries of a table called, NAMTBL, which contains all the legal characters. In the sample program the table has three entries, 'C', 'S', 0. The zero marks the end of the table. If this last entry is reached, then we branch to BACK and a syntax error will eventually result. This table can be expanded to up to 26 letters in any order.

Step III) If a match is found, then we use the index register from the compare loop to get an address from a table of addresses (actually a double table; one for low byte, one for high byte), put the address into a JMP instruction and then execute the JMP. The effect is the same as an indirect JMP.

Step IV) After executing the code which is appropriate to the word, exit through UPDATE, as before.

Following the steps outlined in example two, 24 more reserved words may be easily added.

Listing 2

```
10  00C7=           TXTPTR=$C7
20  00A5=           TOKEN=$A5
30                ;
40  5000            *=$5000
50                ;
60  5000 A001     NEWORD  LDY #1
70  5002 B1C7             LDA (TXTPTR),Y  get 2nd chr of the line
80  5004 C9A5             CMP #TOKEN
90  5006 D054             BNE BACK        if not ster token then RTS
100 5008 88               DEY
110 5009 A200             LDX #0
120 500B BD2450 LOOP      LDA NAMTBL,X
130 500E F04C             BEQ BACK        if et end of teble
140 5010 E8               INX
150 5011 D1C7             CMP (TXTPTR),Y
160 5013 D0F6             BNE LOOP        keep trying if no match
170               ;
180 5015 BD2650 FOUND     LDA LOADR-1,X   get address, lo byte
190 5018 8D2250           STA JMPLO
200 501B BD2850           LDA HIADR-1,X   get address, hi byte
210 501E 8D2350           STA JMPHI
220               ;
230 5022=           JMPLO=*+1
240 5023=           JMPHI=*+2
250               ;
260 5021 4CFFFF           JMP $FFFF       by the time this is execute
270               ;                       the address is changed
280               ;
290 5024 43       NAMTBL  .BYTE 'C','S',0
290 5025 53
290 5026 00
300               ;
310 5027 2E       LOADR   .BYTE $2E,$44
310 5028 44
320               ;
330 5029 50       HIADR   .BYTE $50,$50
330 502A 50
340               ;
350               ;
360               ;code for new reserved word begins here
370               ;
380 502E A920     C.CODE  LDA #32         ASCII for blenk
390 5030 A000             LDY #0          ready for indexed STA
400 502F A208             LDX #8          no. of pages on screen
410 5031 9900D0 PUTIT     STA $D000,Y
420 5034 C8               INY
430 5035 D0FA             BNE PUTIT
440 5037 EE3350           INC PUTIT+2     if Y rolls over then change
450 503A CA               DEX             page
460 503B D0F4             BNE PUTIT
470 503D A9D0             LDA #$D0        restore for
480 503F 8D3350           STA PUTIT+2        next cell
490 5042 D008             BNE UPDATE      elways brenches
500               ;
510               ;
520 5044 AD2A26 S.CODE    LDA 9770
530 5047 4940             EOR #$40
540 5049 8D2A26           STA 9770
550               ;
560               ;need to update TXTPTR before return to BASIC
570               ;
580 504C A5C7     UPDATE  LDA TXTPTR
590 504E 18               CLC
600 504F 6902             ADC #2
620 5053 A5C8             LDA TXTPTR+1
630 5055 6900             ADC #0            add the cerry if it's there
640 5057 85C8             STA TXTPTR+1
650               ;now TXTPTR points to the end-of-line marker
660               ;
670               ;there are two return addresses on the stack
680               ;pull off the top one so thet we return to the
690               ;place where LET wes celled, instead of to LET
700               ;
710 5059 68               PLA
720 505A 68               PLA
730 505B 60               RTS
740               ;
750               ;
760               ;BACK is the mechine code thet was deleted from LET
770               ;and replaced by JSR $5000
780               ;
790 505C 20       BACK    .BYTE $20,$2E,$0F,$85,$96,$84,$97
790 505D 2E
790 505E 0F
790 505F 85
790 5060 96
790 5061 84
790 5062 97
800 5063 60               RTS
810                       .END
```

SOFTWARE FOR OSI

Video Games 1 — $15
Head-On, Tank Battle, Trap!

Video Games 2 — 15
Gremlin Hunt, Indy 5000, Gunfight

Board Games 1 — 15
Cubic, Mini-Gomoku

Dungeon Chase — 10
A real-time, D&D, video game

C1 Shorthand — 12
Two key command entry

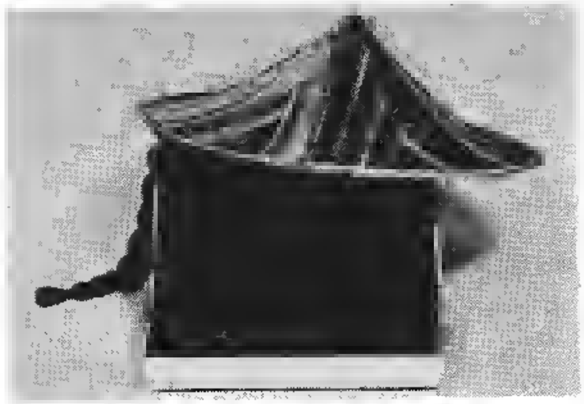
One tape supports all recent ROM systems. Color and sound on video games. Some programs on disk.

FREE CATALOG

Orion Software Associates
147 Main Street
Ossining, NY 10562

WP-6502

a very fine word processor



for OHIO SCIENTIFIC

Tape (C1,C2,C4)	$75	8" 65D & 65U	$125
5" Disk (C1,C2,C4)	$75	Descriptive Brochure	FREE
8" Disk for 65D	$75		

Dwo Quong Fok Lok Sow
23 East 20th Street
New York City, New York 10003
(212)685-2188

Progressive Computing

OHIO SCIENTIFIC

Hardware: C1P VIDEO- gives true 32 or 64 chr/line with guard bands. This is not a make-shift mod. It makes your video every bit as good as the 4P's plus you have switch selectable 1,2 and 3 MHz CPU clock as well as 300, 600 and 1200 baud for cassette and serial port-all crystal controlled. Send for our $1.00 catalog for more details.
COMPLETE PLANS........$18.95 **KIT**.........$39.95
or send your 600 board to:

Rick Lotoczky
3281 Countryside Circle
Pontiac Twp, MI 48057

and he will install the video mod for just $79.95.
OTHER mods available: Add sound-RS-232 port, cassette motor control, cassette interface plans. (300, 600 and 1200 baud)
SOFTWARE: (with documentation) for C1, C2, 4P and 8P. Chess 1.9, Backgammon, Excellent Card Games, Arcade type games, Utility Programs, Mini Word Processor, Memory Maps, Editing Cursor, etc.
Catalog with free program (hard copy) and Memory Map for BASIC in ROM Models......................$1.00
We now accept VISA.

Progressive Computing
3336 Avondale Court
Windsor, Ontario, Canada N9E 1X6
(519) 969-2500

Note: Any modification will VOID your OSI warranty.

MICROCHESS for the AIM

MICRO SOFTWARE

AIM MicroChess with Player's and Programmer's Manual, complete Source Listings, Object on Cassette Tape. $15.00 plus shipping [$1.00 US/$2.00 Anywhere Else]
MICRO Software, P.O. Box 6502, Chelmsford, MA 01824

MICRO CLUB CIRCUIT

N.E.O. Apple Corps

Meets on the third Saturday of each month. Various sub-groups meet during the month. Over 170 members in this fairly new group. They can be contacted at:

N.E.O. Apple Corps
P.O.Box 39364
Cleveland, OH 44139

"Our newsletter was incorrectly listed. If anyone requests information on APPLE BITS they may use the above address. Our primary objectives are to inform area Apple owners of new points of interest and to introduce the community to personal computers through monthly demonstrations."

Wollongong Computer Club

Meets every fortnight and includes a number of small users groups. Presently they are TRS-80, OSI, Pet, Apple, Z80, 8085 and Sorcerer oriented. Address any correspondence to:

Paul Janson
14 Hayward Street
Kanahooka, NSW 2530
Australia

"We also have members with no machine just a common interest."

Appleseed

Meets every other Wednesday at 7:30 p.m. at local computer stores and other locations depending on the program and facilities required. Publish a newsletter. Dr. Terry Mikiten is President. Address inquiries to:

John Ghidoni, Treas.
12801 Huebner Road
San Antonio, TX 78230

"We aim to provide a forum for information exchange, to provide education in the techniques and application of the apple computer by members and outside sources and to provide an interface with other similar clubs throughout the country, specifically including the International Apple Core."

The G.R.A.P.E.
Group for Reilgtous Appte Programming Exchenge

A new international group, and as such they do not have membership meetings. They express their purpose, interest, and activities to be together in a common desire to share their faith and gifts in APPLE programming. They publish a monthly newsletter, The Grape Vine. GRAPE's full policy statement will be sent to all persons expressing an interest by writing to:

G.R.A.P.E.
Stephen M. Lawson
P.O.Box 283
Port Orchard, WA
98366

Permten Besln Amateur Computer Group

Meets on the second Tuesday evening and the second Saturday at different locations. John Rabenaldt is President over 15 members. Several specail interest groups. Write to:

John Rabenaldt
Ector County School District
Box 3912
Odessa, TX 79760

"AIM: To exchange information on small computers and assist in hardware and software projects."

Setem Aree Computer Club Address Chenge

The new address for this club is:

3485 Mock Orange Ct. South
Salem, OR. 97303

PET VET

Loren Wright
P.O. Box 6502
Chelmsford, MA 01824

As the newly appointed "PET expert" on the MICRO staff, I'd like to introduce myself. I have experienced many of the same joys and frustrations you have, from the early lack of documentation to the arrival of the new ROM's. My experience with the PET includes applications to teaching, interfacing peripherals and instruments, hardware modification, character set substitution, and extensive programming in BASIC and machine language.

I will increase my knowledge and experience by constantly reviewing the literature, keeping track of new developments in software, hardware, and firmware, and by strengthening my communication with Commodore. Using the MICRO Lab's PET system, I'll be testing programs and products for the PET, and increasing my own "hands on" experience. As part of MICRO's commitment to the PET, I will become truly an expert. We are aware that many of the PET-oriented magazines are no longer in existence, and it is MICRO's intent to increase our PET coverage to help fill that void.

Meanwhile, I'll be working to expand and improve MICRO's PET coverage. This means printing more PET articles, keeping you informed of new developments, and answering your questions in a "PET VET" column. In the August MICRO we published James Strasma's review of the Programmer's Toolkit, and you can expect that other new PET/CBM hardware, software, and firmware, will be reviewed in future issues.

If you've been wondering where to send that PET/CBM article, this is the place. Send for a MICRO writer's guide if you're having trouble getting started. Also, send in your questions for the **PET VET** column. They can be directed toward any aspect of PET or CBM use.

Finally, remember that there is someone here at MICRO who knows and cares about your PET.

Microbes and Updates

Les Cain found that in "OSI BASIC in ROM, What's Where" (23:65) the five missing keywords can be found by changing line 120 to:

120 FOR C=41062 TO 41089 STEP 3

The program will then include:

AC69	AND
AC66	OR
BAEF	>
ABD8	=
AC96	<

Wendall A. Malpass of Wake Forest, NC. sent the following variations in for some AIM-65 programs:

from 19:38 "Clear"
Location 0305 - LDA 035F
should be: LDA 035F, Y
Location 035F - 43 4C D2
should be: 43 4C 52
Reference to loaded character is at 034E, not 0340.

from 19:39 "Mover"
Location 02C A - 4E 45 D7
should be: 43 4C 57

and from 12:7 "Write to Memory"
If not printing, last line cannot be read. I changed location 0058 to: JSR E993
then, location 000F to: BEQ 005B
0027 to: BEQ 005B
location 005E - "RTS" is preferred over "BRK".

Jerry Tenenbaum of Toronto, Canada, sent in the following information regarding the article "Plotting a Revolution" in 16:5:

On page 8, byte 1E6B should be E2 (not EC)

Loren Wright, MICRO PET Specialist, found the following microbe in "Plotting with Special Character Graphics" 24:11:

On page 13, Figure 1, the second row of symbols was upside down. The whole figure should appear in this order:

Figure 1

Marvin DeJong of Point Lookout, MO found that:

The Morse Code Send/Receive program described in 21:19 will not work if a Mother Plus is attached to the AIM 65. The motherboard's IC U2 prevents any device on the AIM 65 from pulling the IRQ pin low. One solution to get the Send/Receive program to work is to disconnect pin 1 of U2, another solution would be to disconnect the motherboard for this program.

The Mother Plus has recently been redesigned and no longer presents this problem.

A Versatile HI-RES Function Plotter for the APPLE II

One of the obvious usas for APPLE HI-RES capability is to plot various mathematical functions. The program presented hare is very general purposa and parmits the user to simply plot any axprassion as a function of angle from 1 to 360 degrees. A modification is includad which will permit the program to be used on an ATARI as well.

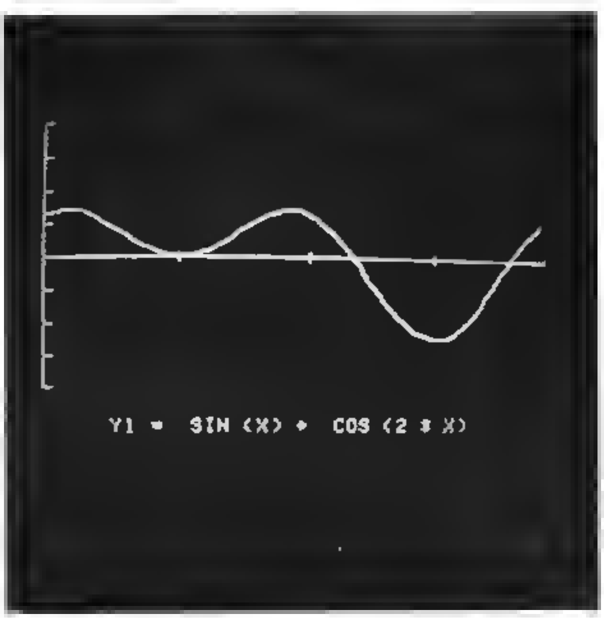
David P. Allan
19 Damon Rd.
Scituata, MA 02066

A few years ago when scientific calculators first made their appearance I was enchanted by the ease with which calculations using transcendental functions could be accomplished. This prompted me to dust off the old trigonometry book and delve into some basics through which I had once passed somewhat painfully. Maybe pain isn't the word. Probably boredom and drudgery would be better words. Log and function tables are probably the only documents with less magnetism than the Little Rock telephone book. I expect that many a budding mathematics curiosity has atrophied over the dryness of log tables.

With the power and freedom of this nifty calculator at hand I suddenly found myself unfettered by the yoke of boredom and I swiftly recovered much of my early curiosity by travelling quickly through basic trigonometry. Gone were the stumbling blocks of look-up tables and I was able to travel down many diversionary "what if's" to see what really happens when certain values change in mathematical formulae.

But as exciting as all this was, and because much of mathematics requires visual images, I looked forward to a time when, with the help of a small computer, I could generate graphs and figures as well as numbers to excite and satisfy my curiosity.

And so it was that after acquiring an Apple II computer, one of my first exercises was to develop a program



which would use Apple's excellent high-resolution graphics to plot the path of a variety of mathematical expressions. This program is the result and I have had much, much fun with it.

The program was developed on an Apple II with 48K of RAM and an Applesoft ROM card. The entire program takes only slightly more than 3K of RAM, depending on the complexity of the function being plotted.

Those who do not have the Applesoft ROM card may still use this program by changing line 480 to read "HGR2" instead of "HGR:" under these circumstances the function plotted formula will not be printed at the bottom of the screen. All other functions work as described.

The heart of the program is line 1010 which contains the function being explored. A typical function is listed here. When run, the program first defines some trigomometric and hyperbolic functions which are not directly available in Applesoft Basic. It then proceeds to plot the X and Y axes. As currently arranged the expression under investigation is plotted as a function of changing angle, from 1 to 360 degrees. By changing lines 670 and 900 other independent variables could be introduced. The program is completely protected against off-scale plotting and automatically scales itself for the range of independent variables selected.

When the plot is completed the program dutifully presents a printout of the function and awaits your pleasure at the push of the return key. It then presents you with a helpful list of all of the additional functions defined by the program in addition to those resident in Applesoft Basic. Line 1010 is listed and the cursor invites your screen editing of this line for further variations.

A word of caution: any attempt to plot mathematical "no-no's" such as square roots or logs of negitive values will earn you a quick error message. Do not despair. Use of the ABS command will quickly get you back in business when these values crop up!

This program has all kinds of tinkering possibilities. You might try surrounding line 1010 with a FOR... Next loop to introduce other variable changes and to allow longer expressions than you can conveniently type into line 1010 all at once. Just beware! This program is subtly laced with a curious narcotic which has been known to keep the user awake all night! Have fun!

μ

```
FIST
100  REM ***********************
110  REM *   FUNCTIØN PLØTTER   *
120  REM *   BY DAVID P. ALLEN  *
130  REM *  (C) CØPYRIGHT 1980  *
140  REM *  APPLESØFT II BASIC  *
150  REM ***********************
160  REM
170  REM
180  REM  THIS PRØGRAM PLØTS A
190  REM  CURVE FØR ANY EXPRESS-
200  REM  IØN AS A FUNCTIØN ØF
210  REM  INCREASING ANGLE FRØM
220  REM  1 TØ 360 DEGREES.
230  REM  CHANGE LINE 1010 TØ A
240  REM  FUNCTIØN YØU WISH TØ
250  REM  PLØT.
260  REM
270  REM
280  REM ** DEFINE FUNCTIØNS **
290  REM
300  DEF  FN SCH(X) = 2 / ( EXP (
X) +  EXP ( - X)): REM  SECH
(X)
310  DEF  FN CCH(X) = 2 / ( EXP (
X) -  EXP ( - X)): REM  CSCH
(X)
320  DEF  FN CTH(X) =  EXP ( - X)
 / ( EXP (X) -  EXP ( - X)) *
2 + 1: REM    CØTH(X)
330  DEF  FN SEC(X) = 1 /  CØS (X
): DEF  FN CSC(X) = 1 /  SIN
(X): DEF  FN CØT(X) = 1 /  TAN
(X)
340  DEF  FN SNH(X) = ( EXP (X) -
 EXP ( - X)) / 2: REM  SINH(
X)
350  DEF  FN CØH(X) = ( EXP (X) +
 EXP ( - X)) / 2: REM  CØSH(
X)
360  DEF  FN TAH(X) =  -  EXP ( -
X) / ( EXP (X) +  EXP ( - X)
) * 2 + 1: REM  TANH(X)
370  REM
380  REM
390  REM ** PLØT GRAPH AXES **
400  REM
410  HØME
```

Courtesy of Roger Wagner's "VAR-DOC"

```
420  REM
430  REM  MØVE CURSØR TØ BØTTØM
440  REM  LINE.
450  REM
460  VTAB 24
470  REM
480  HGR
490  HCØLØR= 7
500  HPLØT 0,80 TØ 279,80
510  HPLØT 0,16 TØ 0,143
520  FØR I = 0 TØ 279 STEP 70
530  HPLØT I,78 TØ I,82: HPLØT 27
9,78 TØ 279,82
540  NEXT I
550  FØR I = 16 TØ 144 STEP 16
560  HPLØT 0,I TØ 4,I
570  NEXT I
580  REM
590  REM  FLAGS FØR FIRST PLØT
600  REM  AND SCALE.
610  REM
620 F = 0:G = 0
630  REM
640  REM  R1 AND R2 MAY BE SET
650  REM  FOR OTHER LIMITS.
660  REM
670 R1 = 1:R2 = 360
680  REM
690  REM
700  REM ** BEGIN PLOT **
710  REM
720  REM  CHANGE STEP FØR MORE
730  REM  ØR LESS RESØLUTIØN.
740  REM  IF R1 > R2 THEN STEP
750  REM  MUST BE NEGATIVE.
760  REM
770  FØR I = R1 TØ R2 STEP 5
780  REM
790  REM  NEXT 3 STEPS ESTABLISH
800  REM  HØRIZØNTAL SCALE.
810  REM
820  IF  ABS (R1) >  =  ABS (R2) THEN
R =  ABS (R1)
830  IF  ABS (R2) >  =  ABS (R1) THEN
R =  ABS (R2)
840  IF G = 0 THEN S = 70 * 4 / R
:G = 1
850 X = I:Y = 0
860  REM
870  REM  CØNVERTS DEGREES TØ
880  REM  RADIANS.
890  REM
900 X = X * 3.14159 / 180
910  REM
920  REM  PREVENTS CRASHING WHEN
930  REM  X = 0.
940  REM
950  IF X = 0 THEN X = .00001
960  REM
970  REM
980  REM  NEXT LINE DESCRIBES
990  REM  FUNCTION TØ BE PLOTTED
1000  REM
1010 Y1 =  SIN (X) +  CØS (2 * X)

1020 Y = Y + Y1
1030 Y = Y * 20
1040  REM
1050  REM  SCALES X
1060  REM
1070 X = I * S
1080  REM
1090  REM  RELATES PLØT TØ X AXIS

1100  REM
1110 Y =  - Y + 80
1120  REM
1130  REM  SUBRØUTINE PREVENTS
1140  REM  OFF-SCALE CRASHING.
1150  REM
1160  GØSUB 1830
1170  REM
1180  REM  PLØTS FIRST PØINT.
1190  REM
1200  IF F = 0 THEN  HPLØT X,Y:F =
1
1210  HPLØT  TØ X,Y
1220  NEXT I
1230  PRINT : LIST 1010
1240  REM
1250  REM  BLANKS ØUT LINE #
1260  REM  AFTER LISTING
1270  REM  LINE 1010.
1280  REM
1290  PØKE 1616,160: PØKE 1617,16
0: PØKE 1618,160: PØKE 1619,
160
1300  REM
1310  REM  WAITING FØR YØUR PLEA-
1320  REM  SURE!  PUNCH 'RETURN'
1330  REM  TØ CONTINUE!
1340  REM
1350  PØKE  - 16368,0: WAIT  - 16
384,128
1360  REM
1370  REM
1380  REM  THRØWS PREVIØUS KEY-
1390  REM  STROKE AWAY WITH
1400  REM  'GET Z$'!
1410  REM
1420  GET Z$
1430  REM
```

```
1430  REM
1440  REM   CLEAR SCREEN AND
1450  REM   PRINT FUNCTIØNS FØR
1460  REM   REMINDER.
1470  REM
1480  TEXT : HØME
1490  PRINT  TAB( 9);"SECANT = FN
 SEC(X)
1500  PRINT  TAB( 9);"CØSEC  = FN
 CSC(X)"
1510  PRINT  TAB( 9);"CØTAN  = FN
 CØT(X)"
1520  PRINT  TAB( 9);"SINH   = FN
 SNH(X)"
1530  PRINT  TAB( 9);"CØSH   = FN
 CØH(X)"
1540  PRINT  TAB( 9);"TANH   = FN
 TAH(X)"
1550  PRINT  TAB( 9);"SECH   = FN
 SCH(X)"
1560  PRINT  TAB( 9);"CSCH   = FN
 CCH(X)"
1570  PRINT  TAB( 9);"CØTH   = FN
 CTH(X)"
1580  REM
1590  REM   NØW WE SET UP LINE
1600  REM   1010 FØR EDITING.
1610  REM   'PØKE 32, 2' MØVES
1620  REM   MARGIN SØ CURSØR CAN
1630  REM   FIT IN FRØNT.
1640  REM
1650  VTAB (12)
1660  PRINT "CHANGE LINE 1010 AS
DESIRED AND"
1670  PRINT "RUN AGAIN!"
1680  PØKE 32,2
1690  LIST 1010
1700  REM
1710  REM   NØW WE RESTØRE MARGIN
1720  REM   AND MØVE CURSØR IN
1730  REM   FRØNT ØF LINE #.
1740  REM
1750  PØKE 32,0
1760  PØKE 37,13: PØKE 36,0
1770  REM
1780  END
1790  REM
1800  REM   SCALE ANTI-CRASHING
1810  REM   SUBRØUTINE.
1820  REM
1830  IF X < 0 THEN X = 0
1840  IF X > 279 THEN X = 279
1850  IF Y < 0 THEN Y = 0
1860  IF Y > 159 THEN Y = 159
1870  RETURN
```

```
******************************
*                            *
*      FUNCTIØN PLØTTER      *
*                            *
*  -->TABLE ØF VARIABLES<--  *
*                            *
******************************
CCH(*) - HYPERBOLIC CØSECANT
310

CØH(*) - HYPERBOLIC CØSINE
350

CØT(*) - CØTANGENT
330

CSC(*) - CØSECANT
330

CTH(*) - HYPERBOLIC CØTANGENT
320

F - FLAG FØR FIRST PLOT
620 1200 1200

G - FLAG FØR SCALE
620 840 840

I - LØØPING VARIABLE
520 530 530 540 550 560 560
570 770 850 1070 1220

R - SCALE FACTOR
820 830 840

R1 - PLØTTING RANGE - START
670 770 820 820 830

R2 - PLØTTING RANGE - END
670 770 820 830 830

S - SCALE
840 1070

SCH(*) - HYPERBOLIC SECANT
300

SEC(*) - SECANT
330

SNH(*) - HYPERBOLIC SINE
340

TAH(*) - HYPERBOLIC TANGENT
360
```

```
X - HØRIZØNTAL PLØTTING VALUE
300 300 300 310 310 310 320
320 320 320 330 330 330 330
330 330 340 340 340 350 350
350 360 360 360 360 850 900
900 950 950 1010 1010 1070
1200 1210 1830 1830 1840 1840

Y - VERTICAL PLØTTING VALUE
850 1020 1020 1030 1030 1110
1110 1200 1210 1850 1850 1860
1860

Y1 - FUNCTIØN VARIABLE
1010 1020

Z$ - KEYSTRØKE USERUPPER
1420

END ØF VAR. LIST
```

David P. Allen is founding partner, chairman of the board and executive producer of the Video Picture Company, Inc., Boston.

His technical background includes consulting engineer for Boston Broadcasters, Inc. to design and build a new VHF facility for channel 5 in Boston. Developed and operated for channel 5 the first electronic news gathering mobile unit in New England.

Senior Engineer, consultant for RCA Corp. in designing educational television facilities.

David Allen's other publications include "Television System Design" for the United States Air Force. He is also a contributing editor for Videography Magazine with monthly production column and other articles.

Here's a bonus for Atari 400 and 800 computer users. I discovered that with only slight modification the function plotter program runs beautifully on Atari 400 with only 8k of memory. The only feature left off from the Apple program is the list of functions, since the Atari basic has no 'FN' command. Atari users would do well to note that contrary to some Atari instructions, there is not'TAN' function in Atari basic. The dilemma fo this absence is easily overcome by using 'SIN' function divided by 'COS' function whereever a target is to be derived. Here is a listing for Atari computers.

μ

```
1 REM FUNCTION PLOTTER PROGRAM
2 REM BY DAVID P. ALLEN
3 REM ATARI FLOATING POINT BASIC
4 REM COPYRIGHT (C) 1980
5 REM
6 REM THIS PROGRAM PLOTS A
7 REM CURVE FOR ANY EXPRESS-
8 REM ION AS A FUNCTION OF
9 REM INCREASING ANGLE FROM
10 REM 1 TO 360 DEGREES.
11 REM CHANGE LINE 2900
12 REM TO A FUNCTION YOU WISH
13 REM TO PLOT.
14 REM
15 REM
40 REM ESTABLISH GRAPH STARTING
41 REM AND ENDING POINTS.
42 REM
43 REM
50 R1=1:R2=360
88 REM
89 REM
90 REM SET GRAPHIC PARAMETERS
91 REM
92 REM
100 GRAPHICS 7
200 COLOR 1
250 SETCOLOR 4,9,4
268 REM
269 REM
270 REM PLOT GRAPH AXIS
271 REM
272 REM
300 PLOT 1,1:DRAWTO 1,80
400 PLOT 1,40:DRAWTO 157,40
500 FOR I=0 TO 80 STEP 10
600 PLOT 1,I:DRAWTO 3,I
700 NEXT I
800 FOR I=1 TO 158 STEP 39
900 PLOT I,38:DRAWTO I,42
1000 NEXT I
1100 REM
1110 REM
1120 REM SET FLAGS FOR FIRST PLOT
1130 REM AND SCALE.
1140 REM
1150 REM
2000 F=0:G=0
2010 REM
2020 REM
2030 REM START PLOTTING
2040 REM
2050 REM
2060 REM CHANGE STEP FOR MORE
2061 REM OR LESS RESOLUTION.
2062 REM IF R1>R2 THEN STEP
2063 REM MUST BE NEGATIVE
2064 REM (PRECEDED BY A MINUS
2065 REM SIGN).
2066 REM
2067 REM
2100 FOR I=R1 TO R2 STEP 3
2110 REM
2120 REM
2130 REM NEXT THREE STEPS ESTABLISH
2140 REM HORIZONTAL SCALE.
2150 REM
2160 REM
2200 IF ABS(R1)>=ABS(R2) THEN R=ABS(R1
2300 IF ABS(R2)>=ABS(R1) THEN R=ABS(R2
2400 IF G=0 THEN S=158/R:G=1
2500 X=I:Y=0
2550 REM
2551 REM
2552 REM CONVERT DEGREES TO
2553 REM RADIANS.
2554 REM
2555 REM
2600 X=X*3.14159/180
2650 REM
2651 REM
2652 REM PREVENTS CRASHING WHEN
2653 REM X=0.
2654 REM
2655 REM
2800 IF X=0 THEN X=1.0E-05
2850 REM
2851 REM
2852 REM NEXT LINE DESCRIBES
2853 REM FUNCTION TO BE PLOTTED.
2854 REM
2855 REM
2900 Y1=SIN(X)*COS(X^2)
3000 Y=Y+Y1
3100 Y=Y*20
3150 REM
3151 REM
3152 REM SCALES X.
3153 REM
3154 REM
3200 X=I*S
3250 REM
3251 REM
3252 REM RELATES PLOT TO X AXIS.
3253 REM
3254 REM
3300 Y=-Y+40
3350 REM
3351 REM
3352 REM SUBROUTINE PREVENTS
3353 REM OFF-SCALE CRASHING.
3354 REM
3355 REM
3400 GOSUB 5000
3450 REM
3451 REM
3452 REM PLOTS FIRST POINT.
3453 REM
3454 REM
3500 IF F=0 THEN PLOT X,Y:F=1
3600 DRAWTO X,Y
3700 NEXT I
3750 REM
3751 REM
3752 REM DISPLAYS EQUATION OF
3753 REM PLOTTED FUNCTION BENEATH
3754 REM GRAPHIC DISPLAY.
3755 REM
3756 REM
3800 LIST 2900
3900 END
5000 IF X<0 THEN X=0
5100 IF X>158 THEN X=158
5200 IF Y<0 THEN Y=0
5300 IF Y>80 THEN Y=80
5400 RETURN
```

```
1 REM FUNCTION PLOTTER PROGRAM
2 REM BY DAVID P. ALLEN
3 REM ATARI FLOATING POINT BASIC
4 REM COPYRIGHT (C) 1980
5 REM
6 REM THIS PROGRAM PLOTS A
7 REM CURVE FOR ANY EXPRESS-
8 REM ION AS A FUNCTION OF
9 REM INCREASING ANGLE FROM
10 REM 1 TO 360 DEGREES.
11 REM CHANGE LINE 2900
12 REM TO A FUNCTION YOU WISH
13 REM TO PLOT.
14 REM
15 REM
16 REM GRAPHICS 8 VERSION
17 REM REQUIRES MINIMUM OF
18 REM 16K RAM MEMORY.
19 REM
20 REM
40 REM ESTABLISH GRAPH STARTING
41 REM AND ENDING POINTS.
42 REM
43 REM
50 R1=1:R2=360
88 REM
89 REM
```

```
90 REM SET GRAPHIC PARAMETERS
91 REM
92 REM
100 GRAPHICS 8
200 COLOR 1
250 SETCOLOR 4,9,4
268 REM
269 REM
270 REM PLOT GRAPH AXIS
271 REM
272 REM
300 PLOT 1,1:DRAWTO 1,160
400 PLOT 1,80:DRAWTO 314,80
500 FOR I=0 TO 160 STEP 19.9
600 PLOT 1,I:DRAWTO 6,I
700 NEXT I
800 FOR I=1 TO 314 STEP 78
900 PLOT I,76:DRAWTO I,84
1000 NEXT I
1100 REM
1110 REM
1120 REM SET FLAGS FOR FIRST PLOT
1130 REM AND SCALE.
1140 REM
1150 REM
2000 F=0:G=0
2010 REM
2020 REM
2030 REM START PLOTTING
2040 REM
2050 REM
2060 REM CHANGE STEP FOR MORE
2061 REM OR LESS RESOLUTION.
2062 REM IF R1>R2 THEN STEP
2063 REM MUST BE NEGATIVE
2064 REM (PRECEDED BY A MINUS
2065 REM SIGN).
2066 REM
2067 REM
2100 FOR I=R1 TO R2 STEP 3
2110 REM
2120 REM
2130 REM NEXT THREE STEPS ESTABLISH
2140 REM HORIZONTAL SCALE.
2150 REM
2160 REM
2200 IF ABS(R1)>=ABS(R2) THEN R=ABS(R1)
2300 IF ABS(R2)>=ABS(R1) THEN R=ABS(R2)
2400 IF G=0 THEN S=314/R:G=1
2500 X=I:Y=0
2550 REM
2551 REM
2552 REM CONVERT DEGREES TO
2553 REM RADIANS.
2554 REM
2555 REM
2600 X=X*3.14159/180
2650 REM
2651 REM
2652 REM PREVENTS CRASHING WHEN
2653 REM X=0.
2654 REM
2655 REM
2800 IF X=0 THEN X=1.0E-05
2850 REM
2851 REM
2852 REM NEXT LINE DESCRIBES
2853 REM FUNCTION TO BE PLOTTED.
2854 REM
2855 REM
2900 Y1=SIN(X)*COS(X^2)
3000 Y=Y+Y1
3100 Y=Y*20
3150 REM
3151 REM
3152 REM SCALES X.
3153 REM
3154 REM
3200 X=I*S
3250 REM
3251 REM
3252 REM RELATES PLOT TO X AXIS.
3253 REM
3254 REM
3300 Y=-Y+80
3350 REM
3351 REM
3352 REM SUBROUTINE PREVENTS
3353 REM OFF-SCALE CRASHING.
3354 REM
3355 REM
3400 GOSUB 5000
3450 REM
3451 REM
3452 REM PLOTS FIRST POINT.
3453 REM
3454 REM
3500 IF F=0 THEN PLOT X,Y:F=1
3600 DRAWTO X,Y
3700 NEXT I
3750 REM
3751 REM
3752 REM DISPLAYS EQUATION OF
3753 REM PLOTTED FUNCTION BENEATH
3754 REM GRAPHIC DISPLAY.
3755 REM
3756 REM
3800 LIST 2900
3900 END
5000 IF X<0 THEN X=0
5100 IF X>314 THEN X=314
5200 IF Y<0 THEN Y=0
5300 IF Y>160 THEN Y=160
5400 RETURN
```

MICRO SCOPE

Number 2

KIM — VENTURE ™

1. **Microcomputers which can use product:** KIM-1
2. **System hardware requirements:** The only requirements are a standard 1K KIM-1 and a Phillips-type cassette tape recorder.
3. **System software requirements:** None.
4. **Product features:** KIM-VENTURE is a fantasy-logic game of the "dungeons and dragons" genre. The objective of the game is to negotiate through a complex maze, find the hidden treasures, and return home with them. The KIM's keypad is used to direct movements and to manipulate the environment, e.g., picking up treasure and various tools along the route. Feedback in the form of present location, hazards, available tools, etc. is provided by cryptic messages written in the KIM alphabet (see "The First Book of KIM").
5. **Product performance:** KIM-VENTURE is programmed in three parts which must be loaded separately. Of the dozen or so times that all three sections were loaded, not a single load error was encountered. The program is well thought through. For example, the LED messages take some time for the inexperienced user to decipher, but provisions have been made to allow the user to lengthen the display time, or to have the messages repeated. The cryptic nature of the display is certainly not a liability. Deciphering the display adds to the mystery and fantasy of the game. Like most "dungeons and-dragons" games, KIM-VENTURE has multiple levels of play. As the player gains experience, he discovers new moves and exciting new possibilities to be explored. In short, KIM-VENTURE performs as advertised.
6. **Product quality:** KIM-VENTURE is a well written and very efficient machine language program. It is hard to believe that this program fits into 1K.
7. **Product limitations:** Not applicable.
8. **Product documentation:** The instructions for loading and playing the game are clearly and completely described. In addition, a complete source listing of the software is provided and is annotated in detail, so that the program can be traced with little difficulty. For the impatient and faithless, the complete solution to the KIM-VENTURE maze is also provided.
9. **Special user requirements:** Other than being able to load a KIM program, there are no special user requirements.
10. **Price/Feature/Quality evaluation:** Priced at $24.95, KIM-VENTURE is an expensive piece of software; however given the relativly small market for entertainment software for the KIM, the costs of developing this type of software, and the high quality of this package, the tradeoffs are fair.
(Ed's Note: Mr. Leedom will be distributing this program himself and has asked that we mention that he is now able to reduce the price significantly to $14.95. This 40 % decrease should increase the tradeoff value. To order simply send to the author Robert C. Leedom, 14069 Stevens Valley Ct., Glenwood, MD 21738)
11. **Additional comments:** If you become impatient with problems that take more than a few minutes to solve, or have no understanding of the autistic pleasures of a good puzzle, the KIM-VENTURE would be a poor investment. If, on the other hand, you savor the challenge of solving complex problems, KIM-VENTURE could be a cheap investment, measured in terms of costs per hour of entertainment.
(Ed's note: One major feature of the product which is not mentioned but might be of value; KIM-Venture comes with a fully-documented scoring program which is loaded and run when the game is finished. The scoring program then rates you as having achieved one of eleven levels of skill, and shows you how many moves it took you to get to that level. This allows competition between many players by comparing scores.)
12. **Reviewer:** Dr. Mark H. Meinrath, c/o A.H. Meinrath, 302 Dolphin Place, Corpus Christi, TX 78411

Tiny PILOT for the AIM

Tiny PILOT is e compect progremming lenguege which cen edd e lot of versetility to your microcomputer. This version hes been developed to run on the AIM 65. It is e very inexpensive wey to edd higher level lenguege cepebility to your system.

Larry Kollar and Carl Gutekunst
257 W. Wadsworth Hall
Michigan Technological University
Houghton, MI 49931

Nicholas Vrtis' article "Tiny Pilot" (MICRO 16:41) shows that good things still come in small packages. However, a few routines, the editor in particular, can be deleted or replaced when implementing the interpreter with monitor routines on the AIM-65. After tearing the program down, Carl and I finally found the last place needing a CMP -CR inserted and we had enough room left over for two more instructions and a startup message.

The AIM PILOT program is mainly built on the framework of Mr. Vrtis' program, with some small changes to accommodate the new instructions. The first of these, P:ON or P:OFF, simply turns the printer on to off accordingly. To check which way the printer is to be switched, the second letter following the colon is looked at. If this letter is an 'N', the PRIFLG ($A411) byte is set to $80. Anything else is assumed to be P:OFF and $00 is stored in PRIFLG. The remaining letters are then skipped and the next instruction is fetched.

The next instruction, H:ADDR, is a bit more complicated. This instruction calls a machine language subroutine at the hexadecimal address ADDR then returns to the main program. Fortunately, the monitor routines HEX and PACK do the ASCII to binary conversions easily. The resulting byte is stored in the page zero locations called HEXSUB. An indirect subroutine call is simulated by calling an indirect jump then advancing to the next instruction. H: can be used to escape the limitations of a 768-byte interpreter by adding one's own functions such as multiply routines or random number generators. Computation never was PILOT's strong point...

Obviously, this program will not run on a 1K AIM. Also, entering source code would have been much nicer if Issue 19:37 (HEX LOAD) hand loading the netire gets rather tiring, considering that it took about eight trys to make PILOT run on the AIM.

To enter PILOT text on the AIM, use the text editor like always, entering 0500, space, space; and begin writing. I have the F1 function key set aside to run the interpreter; * = 0200, G, space will serve as well. The interpreter displays its "signon" message, then executes the program. Some final cautions: there are no diagnostics or actual error messages, so debugging can be difficult. On the other hand, PILOT is such an easy language that it would be hard to make a subtle mistake. P: is foolproof enough, but I would recommend using the full address field for the **H**: routine (four hexadecimal characters).

PILOT is an economical language, both in terms of space and cost. I would not throw the $100 for the BASIC chips unless I had a video monitor (more money), and the few places Tiny PILOT falls down can be easily worked around. The language is easy to learn, so give it a try.

μ

References

1. Tiny PILOT: An Educational Language for the 6502, Nicholas Vrtis, Micro 16:41.

2. Sharpen your AIM, Robert E. Babcock, Micro 19:37.

```
*****  AIM PILOT - 2R1M1A ******

@6502*CROSS ASM.SG 6502*6502 PILOT
6502*CROSS  IR3M5 74R1N2 03/04/80 18 57 09 (44)
0000            ;****************************************************************
0000            ;
0000            ; Tiny PILOT  An educational language for the 6502
0000            ;
0000            ; By Nicholas Vrtis. 5863 Pinetree S E . Kentwood. MI 49508
0000            ;
0000            ; Modified for the AIM-65 Microcomputer System
0000            ; by Larry E. Kollar and Carl S. Gutekunst
0000            ;    257 W Wadsworth. Michigan Technological University
0000            ;    Houghton. MI 49931
0000            ;
0000            ; March 1980
0000            ;
0000            ;****************************************************************
0000            ;
0000            ; FORMAT * STATEMENT              * WHAT IT DOES
0000            ;****************************************************************
0000            ; T TEXT * TYPE                   * Display text on D/P
0000            ; A      * ACCEPT                 * Input characters into ANSWER field
0000            ; ?      * ACCEPT NAME            * Input characters into NAME and
0000            ;        *                        *   ANSWER fields
0000            ; M TEXT * MATCH TEXT             * Compare TEXT to ANSWER field and
0000            ;        *                        *   set MATCH FLAG to  Y  if equal.
0000            ;        *                        *     N  if not
0000            ; J N    * JUMP                   * JUMP to label N
0000            ; J A    * JUMP ACCEPT            * JUMP back to last ACCEPT statement
0000            ; J *    * JUMP RESTART           * Restart program from the beginning
0000            ; U N    * USE SUBROUTINE         * Call subroutine at label N
0000            ; E      * EXIT FROM SUBROUTINE   * Return from last subroutine
0000            ; S      * STOP                   * Stop program and return to AIM monitor
0000            ; P ON   * PRINTER CONTROL        * Turn printer ON/OFF
0000            ; P OFF  *
0000            ; H ADDR * HEX SUBROUTINE         * Call a machine language
0000            ;        *                        *   subroutine at adress ADDR
0000            ; C      * COMPUTE                * Perform arithmetic on variables named
0000            ;        *                        *   A through Z  Only + and - operations
0000            ;        *                        *   allowed. range is -999 to +999
0000            ;        *                        *   $= places result in ANSWER field
0000            ; R      * REMARK                 * Comments. remarks. etc
0000            ; *N     * LABEL                  * Destination for JUMP or USE. may preceed
0000            ;        *                        *   and statement or conditional
0000            ; $X     * VARIABLE ITEM          * When used in TEXT allows contents of
0000            ;        *                        *   variable X to be displayed or matched
0000            ;        *                        *   $? indicates NAME field
0000            ;        *                        *
0000            ;        * CONDITIONALS           * May preceed any statement
0000            ; N      *                        * Execute only if MATCH FLAG is 'N'
0000            ; Y      *                        * Execute only if MATCH FLAG is 'Y'
0000            ;        *                        *
0000            ;****************************************************************
0000            ;
0000            ;       External Subroutines
0000            ;
0000            GM      = $E7D4         ; Output a '?' to the display
0000            HEX     = $EA7D         ; Change A from ASCII to binary in LSD of A
0000            RACK    = $EA84         ; Move last call to HEX into MSB and call HEX
0000            CRLF    = $E9F0         ; Output a <CR> and <LF>
****** AIM PILOT - 2R1M1A ******

0000            START   = $E182         ; AIM Monitor Warm Start
0000            REDOUT  = $E973         ; Input one character with BS allowed
0000            OUTALL  = $E9BC         ; Output one character
0000            ;
0000            ;       Page Zero Locations
0000            ;
0000            *       = $0000
0000            LST     * = *+2         ; Address of last ACCEPT command
0002            FLG     * = *+1         ; Current Y/N flag
0003            CHRS    * = *+60        ; 60 byte input buffer
003F            IFLAG   * = *+1         ; Special indicator flag area
0040            HOLDY   * = *+1         ; Hold area for Y value
0041            WORK    * = *+2         ; Temp work variable
0043            RESULT  * = *+2         ; Result hold area for computations
0045            ANSX    * = *+1         ; Hold area for ANSWER index pointer
0046            SIGNIF  * = *+1         ; Significance indicator
0047            NAME    * = *+(2*26)    ; Variables - 2 bytes each
007B            VARIBS  * = *+(2*26)    ; Variables - 2 bytes each
```

```
00AF            OPRATN * = *+1          ; Last operation in compute statement
00B0            NUMDSP * = *+5          ; Display variable build area
00B5            RETURN * = *+2          ; USE return address
00B7            CURAB  * = *+2          ; Address of current line
00B9            HEXSUB * = *+2          ; Address of machine language subroutines
00BB            ;
00BB            ;       Constants
00BB            ;
00BB            CR     = $0D            ; Carriage Return
00BB            BS     = $7F            ; Delete
00BB            ;
00BB            ;
00BB            ;       External location
00BB            ;
00BB            PRIFLG = $A411          ; Printer control flag
00BB            ;
00BB            ;       PILOT Interpreter start
00BB            ;
00BB            *  = $0200
0200            ;
0200 20F0E9     EXEC   JSR  CRLF        ; Clear D/P
0203            ;
0203 206704     RESTRT JSR  SETBGN      ; Come here from J.*
0206 200704            JSR  PRT         ; Go and print signon message
0209 A233              LDX  #$33        ; Zero variable areas
020B A900              LDA  #$00
020D 85B6              STA  RETURN+1
020F 957B       RESTR1 STA  VARIBS,X
0211 CA                DEX
0212 10FB              BPL  RESTR1
0214            ;
0214 B1B7       LSTART LDA  (CURAD),Y   ; Get character from line
0216 C92A              CMP  #'*'        ; Label?
0218 D004              BNE  CHKCON      ; No
021A C8                INY              ; Increment past *
021B C8         SKPNXT INY              ;   and next character
021C D0F6              BNE  LSTART      ; Unconditional
021E            ;
021E            ;       CHECK CONDITIONALS
021E            ;
021E C959       CHKCON CMP  #'Y'        ; Check for Y request
0220 F004              BEQ  TFLAG
****** AIM PILOT - 2R1M1A ******

0222 C94E              CMP  #'N'        ; If not. check for N request
0224 D009              BNE  STRTST      ; Branch if neither
0226            ;
0226 C502       TFLAG  CMP  FLG         ; Does request match flag?
0228 F0F1              BEQ  SKPNXT      ; Yes - skip to next char and execute line
022A            ;
022A 204004     FWD    JSR  FWD1        ; No match - skip this statement
022D B0E5              BCS  LSTART      ; Unconditional
022F            ;
022F            ;       START PROCESSING OF LINE
022F            ;
022F 853F       STRTST STA  IFLAG
0231 C8                INY              ; Increment past ':'
0232 C8                INY              ; and point to next char
0233            ;
0233            ;       ?: ENTER NAME STATEMENT
0233            ;
0233 C93F       XQUEST CMP  #'?'        ; Shall we?
0235 D005              BNE  XA          ; Nah.
0237 38                SEC              ; Turn on high order bit to flag it
0238 663F              ROR  IFLAG       ; Processing NAME command
023A D00C              BNE  TAKEIN      ; Now use the ACCEPT logic
023C            ;
023C            ;       ACCEPT STATEMENT
023C            ;
023C C941       XA     CMP  #'A'        ; See if we have ACCEPT statement
023E D030              BNE  XC          ; No. try the COMPUTE
0240 A5B7              LDA  CURAB       ; Save address of the 'A' statement
0242 8500              STA  LST         ; NOTE:  Will include conditionals
0244 A5B8              LDA  CURAB+1
0246 8501              STA  LST+1
0248            ;
0248 20D4E7     TAKEIN JSR  GM          ; Put a '?' to the display
024B A23B              LDX  #$3B        ; Chrs get stored backwards
024D 2073E9     ACHR   JSR  REDOUT      ; Get an input character
0250 C97F              CMP  #BS         ; Is it a backspace?
0252 D007              BNE  ACHR1       ; No. set it in
```

```
0254 E03B           CPX  #59          ; Already at beginning of line?
0256 F0F5           BEQ  ACHR         ; Yes, ignore BS completely
0258 E8             INX               ; Else forget about last character
0259 D0F2           BNE  ACHR         ; Unconditional branch back
025B 9503    ACHR1  STA  CHRS,X       ; And save it for match statement
025D 243F           BIT  IFLAG        ; See if we have NAME field
025F 1002           BPL  ACHR2        ; No, don't store it
0261 9547           STA  NAME,X       ; Else save it in the NAME field too
0263 C90D    ACHR2  CMP  #CR          ; Is it done yet?
0265 F003           BEQ  ADONE        ; Yes, so clear the display
0267 CA             DEX               ; Else advance for next input
0268 10E3           BPL  ACHR         ; And so get it if there's still room
026A 20F0E9  ADONE  JSR  CRLF         ; Got it all, let 'em know
026B 4C2A02         JMP  FWB
0270        ;
0270        ;       COMPUTE STATEMENT
0270        ;
0270 C943    XC     CMP  #'C'         ; Is it 'C' for COMPUTE?
0272 F003           BEQ  XC1          ; Yes, hop over the JMP
0274 4C0203         JMP  XM           ; Else long jump to test for M
0277 207C04  XC1    JSR  GET1BX       ; Get index pointer to RESULT
027A 8645           STX  ANSX         ; Save it for now
027C A900           LDA  #$00         ; Clear RESULT
****** AIM PILOT - 2R1M1A ******

027E 8543           STA  RESULT
0280 8544           STA  RESULT+1
0282 C8             INY               ; Point to '='
0283 A22B           LDX  #'+'         ; Set 1st operation to '+' for add
0285 D04A           BNE  OPWRAP       ; Go save and set up work area
0287        ;
0287        ;       LOOP FOR EACH NEW CHARACTER IN COMPUTE PROCESSING
0287        ;
0287 C8      CMPLOP INY               ; Bump to next character
0288 B1B7           LDA  (CURAD),Y    ; Get said character
028A 3020           BMI  ISOPR        ; Minus is Delete, also last 'Operator'
028C C92F           CMP  #'/'         ; Is it '/' for an operation specified?
028E 901C           BCC  ISOPR        ; Branch if yes
0290 C93A           CMP  #':'         ; If not--is it ':' for a number?
0292 D012           BCS  NOTNMB       ; Branch if not--must be a variable
0294 290F           AND  #$0F         ; Convert number to binary
0296 6A             ROR  A            ; Shift to high order part of A
0297 6A             ROR  A
0298 6A             ROR  A
0299 6A             ROR  A            ; Leave bit 3 in Carry
029A A204           LDX  #$04         ; 4 bits to roll into work
029C 2642    BITROL ROL  WORK+1       ; Ripple carry into WORK
029E 2641           ROL  WORK         ; For 16 bits
02A0 0A             ASL  A            ; Put next bit into carry
02A1 CA             DEX               ; Count one just done
02A2 D0F8           BNE  BITROL       ; Continue if more to go
02A4 F0E1           BEQ  CMPLOP       ; Else get next character (digits)
02A6        ;
02A6 208404  NOTNMB JSR  VTRANS       ; Transfer variable to work area
02A9 4C8702         JMP  CMPLOP       ; Go get next character or operation
02AC        ;
02AC        ;
02AC        ;
02AC F8      ISOPR  SED               ; Set to decimal mode
02AD AA             TAX               ; Save new operation in X for now
02AE A5AF           LDA  OPRATN       ; Get previous operation
02B0 C92D           CMP  #'-'         ; Was it a '-' for Subtract?
02B2 F010           BEQ  OPMNUS       ; Yes, branch forward
02B4 18             CLC               ; All others assume it is add
02B5 A542           LDA  WORK+1
02B7 6544           ADC  RESULT+1
02B9 8544           STA  RESULT+1
02BB A541           LDA  WORK
02BD 6543           ADC  RESULT
02BF 8543           STA  RESULT
02C1 4CD102         JMP  OPWRAP       ; Do wrap up the operation
02C4        ;
02C4 38      OPMNUS SEC               ; Subtraction
02C5 A544           LDA  RESULT+1
02C7 E542           SBC  WORK+1
02C9 8544           STA  RESULT+1
02CB A543           LDA  RESULT
02CD E541           SBC  WORK
02CF 8543           STA  RESULT
02D1        ;
02D1 D8      OPWRAP CLD               ; Get out of decimal mode
02D2 86AF           STX  OPRATN       ; Save new operation
02D4 E00D           CPX  #CR          ; End of line?
02D6 F008           BEQ  CMPDON       ; Yes, we're done
02D8 A900           LDA  #$00         ; Else clear work area for next one

****** AIM PILOT - 2R1M1A ******

02DA 8541           STA  WORK
02DC 8542           STA  WORK+1
02DE F0A7           BEQ  CMPLOP       ; And so do next character
02E0        ;
02E0 A645    CMPDON LDX  ANSX         ; Get index to result
02E2 1013           BPL  TOVRIB       ; Plus is normal index to a variable
02E4 A238           LDX  #$38         ; Else fudge index for 'FROM' result
                                      ;     Using 'RESULT--VARIBS'
02E6        ;
02E6 20870C         JSR  VTRANS+3     ; Move result to work area
02E9 20930C         JSR  CNVDSP+3     ; Convert it to display form
02EC A204           LDX  #$04         ; Transfer display to answer area
02EE B5B0    TALOOP LDA  NUMDSP,X
02F0 951A           STA  CHRS+23,X    ; Note offset to put it at the end
02F2 CA             DEX
02F3 10F9           BPL  TALOOP
02F5 3008           BMI  XFWD         ; Unconditional
02F7 A544    TOVRIB LDA  RESULT+1     ; Desired value
02F9 957C           STA  VARIBS+1,X
02FB A543           LDA  RESULT
02FD 957B           STA  VARIBS,X
02FF 4C2A0   XFWD   JMP  FWD          ; And so do next one
0302        ;
0302        ;       PROCESS MATCH STATEMENT
0302        ;
0302 C94D    XM     CMP  #'M'         ; Is it 'M' for MATCH?
0304 D053           BNE  XU           ; No, try the USE
0306 88             DEY               ; Back up one for what follows
0307 C8      MCHKX  INY               ; Point to match character
0308 A23B           LDX  #$3B         ; Start at 1st accepted character
030A B1B7    MCHK   LDA  (CURAD),Y    ; Get the match character
030C C90D           CMP  #CR          ; End of line yet?
030E F008           BEQ  MXY          ; They have matched to end of 'M:' stmt
0310 D503           CMP  CHRS,X       ; Check for match
0312 D008           BNE  MXNMCH       ; Branch if match failed
0314 C8             INY               ; Else bump to next pair of characters
0315 CA             DEX
0316 10F2           BPL  MCHK         ; And so check if still data left
0318 A259    MXY    LDX  #'Y'         ; Both equal--set flag to 'Y'
031A D039           BNE  MX           ; Unconditional
031C C924    MXNMCH CMP  #'$'         ; Is it '$' for Variable request?
031E F015           BEQ  MNUMB        ; Yes--match to numeric variable
0320 C92C           CMP  #','         ; Is it a comma group separator?
0322 F0F4           BEQ  MXY          ; Yes--matched so far--set it as yes
0324        ;
0324 C8      MCOMMA INY               ; No--so need to skip ahead to comma
0325 B1B7           LDA  (CURAD),Y
0327 C90D           CMP  #CR          ; EOL?
0329 F028           BEQ  MXSETN       ; If to EOL, there is no more to check
032B C92C           CMP  #$2C         ; Check for a comma character
032B F0DB           BEQ  MCHKX        ; Restart compare at next match character
032F D0F3           BNE  MCOMMA       ; Loop in search of a comma
0331 A440    MCOMX  LDY  HOLDY        ; Reset Y to current line pointer
0333 D0EF           BNE  MCOMMA       ; And so look for next comma
0335        ;
0335 C8      MNUMB  INY               ; Variable--bump to variable ID
0336 8645           STX  ANSX         ; Save current X for now
0338 20900C         JSR  CNVDSP       ; Convert variable to display form
033B A645           LDX  ANSX         ; Get pointer to input back
033D 8440           STY  HOLDY        ; Save current 'Y' pointer
033F A004           LDY  #$04         ; Have to search up to 5 bytes
****** AIM PILOT - 2R1M1A ******

0341        ;
0341 B9B000  MXNOLP LDA  NUMDSP,Y     ; Get one numeric character
0344 F008           BEQ  MXDIFF       ; Branch if end--might be match
0346 D503           CMP  CHRS,X       ; Else check against input
0348 D0E7           BNE  MCOMX        ; Branch if no match
034A CA             DEX               ; Else continue matching
034D 88             DEY
034C 10F3           BPL  MXNOLP       ; Unconditional loop return
034E        ;
034E A440    MXDIFF LDY  HOLDY        ; Reset Y to current line pointer
0350 C8             INY               ; Bump to character after variable
```

```
0351 D0B7              BNE  MCHK              ; Unconditional continue checking
0353                .
0353 A24E    MXSETN LDX  #'N'              ; Get 'N'--match was unsuccessful
0355 8602    MX     STX  FLG               ; Store it
0357 D0A6           BNE  XFWD              ; Unconditional forward to next line
0359                .
0359                ;      PROCESS USE SUBROUTINE STATEMENT
0359                ;
0359 C955    XU     CMP  #'U'              ; Is it a 'U' for Use Subroutine?
035B D011           BNE  XJ                ; Branch if not
035D B1B7           LDA  (CURAD),Y         ; Get destination
035F 48             PHA                    ; Save the label character
0360 204004         JSR  FWD1              ; Move to start of next line
0363 A5B7           LDA  CURAD
0365 85B5           STA  RETURN            ; Save for return address
0367 A5B8           LDA  CURAD+1
0369 85B6           STA  RETURN+1
036B 68             PLA                    ; Get destination back
036C D006           BNE  JDO               ; Now go handle as JUMP statement
036E                .
036E                .      PROCESS JUMP STATEMENT
036E                ;
036E C94A    XJ     CMP  #'J'              ; Is it 'J' for JUMP statement?
0370 D02F           BNE  XS                ; Branch if not
0372 B1B7           LDA  (CURAD),Y         ; Get destination
0374                ;
0374 853F    JDO    STA  IFLAG             ; Save label character
0376 C92A           CMP  #'*'              ; Have '*' to request return to beginning?
0378 F024           BEQ  IREST             ; Yes, branch back to top
037A C941           CMP  #'A'              ; See if a labelled jump
037C D00D           BNE  JF                ; If not 'A', it's a normal jump
037E A500           LDA  LST               ; Else set to start of last accept
0380 85D7           STA  CURAD
0382 A501           LDA  LST+1
0384 85B8           STA  CURAD+1
0386 4C0104         JMP  ILNEXT            ; Unconditional
0389                ;
0389 206A04  JF     JSR  SETBGN+3          ; And set back to start of program
038C                .
038C B1D7    FNDMRK LDA  (CURAD),Y         ; Get 1st character
038E C92A           CMP  #'*'              ; Is it '*' for a marker?
0390 D007           BNE  FMNEXT            ; Nope--go ahead to next line
0392 C8             INY                    ; Else bump to marker character
0393 B1D7           LDA  (CURAD),Y         ; Get label
0395 C53F           CMP  IFLAG             ; See if it's the one we want
0397 F068           BEQ  ILNEXT            ; Yes--go execute it
0399 204004  FMNEXT JSR  FWD1              ; Else go to next line
039C B0EE           BCS  FNDMRK            ; And continue looking
039E 4C0302  IREST  JMP  RESTRT            ; Indirect to RESTRT

****** AIM PILOT - 2R1M1A ******

03A1                .
03A1                .      STOP STATEMENT
03A1                .
03A1 C953    XS     CMP  #'S'              ; Is it an 'S' for STOP?
03A3 D003           BNE  XE                ; No, try Exit
03A5 4C82E1         JMP  START             ; Yes, jump into AIM Monitor
03A8                .
03A8                .      EXIT FROM SUBROUTINE
03A8                .
03A8 C945    XE     CMP  #'E'              ; Is it an 'E' for EXIT?
03AA D010           BNE  XR                ; No, try Remark
03AC A5B6           LDA  RETURN+1          ; Move return address to CURAD
03AE F010           BEQ  XXFWD             ; Skip line if not set
03B0 85B8           STA  CURAD+1
03B2 A5B5           LDA  RETURN
03B4 85B7           STA  CURAD
03B6 A900           LDA  #$00              ; Now set to not-used again
03B8 85B6           STA  RETURN+1
03BA F045           BEQ  ILNEXT            ; Unconditional
03BC                .
03BC                .      REMARK STATEMENT
03BC                .
03BC C952    XR     CMP  #'R'              ; Is it an 'R' for REMARK?
03BE D003           BNE  XP                ; Branch if not--else skip the line
03C0 4C2A02  XXFWD  JMP  FWD               ; Can't reach that far alone
03C3                .
03C3                . PRINTER CONTROL--ON/OFF
03C3                .
```

```
03C3 C950    XP     CMP  #'P'              ; Printer control?
03C5 D011           BNE  XH                ; No, try the HEX command
03C7 C8             INY                    ; Both ON and OFF start with O
03C8 B1B7           LDA  (CURAD),Y         ; Get second character
03CA A200           LDX  #$00              ; Default printer OFF
03CC C94E           CMP  #'N'              ; Want printer ON?
03CE D002           BNE  PSTORE
03D0 A280           LDX  #$80              ; For printer = ON
03D2 8E11A4  PSTORE STX  PRIFLG
03D5 4C2A02  FEND   JMP  FWD               ; And get back to work
03D8                .
03D8                . CALL MACHINE LANGUAGE SUBROUTINE
03D8                .
03D8 C948    XH     CMP  #'H'              ; Is it an 'H'?
03DA D01C           BNE  XT                ; No, go to TYPE
03DC A202           LDX  #$02              ; Do this twice
03DE B1B7    XADD   LDA  (CURAD),Y         ; Get first half of byte
03E0 207DEA         JSR  HEX               ; Clear A and convert to binary
03E3 C8             INY                    ; Next part--complete byte
03E4 B1B7           LDA  (CURAD),Y
03E6 2084EA         JSR  PACK              ; Form the high order address
03E9 C8             INY                    ; Set up for next character
03EA CA             DEX
03EB 95B9           STA  HEXSUB,X          ; Store it
03ED D0EF           BNE  XADD              ; No, get next byte
03EF 20F503         JSR  XGO
03F2 4C2A02         JMP  FWD
03F5 6C8900  XGO    JMP  (HEXSUB)          ; And away....
03F8                .
03F8                .      TYPE STATEMENTS AND SYNTAX ERRORS
03F8                ;
03F8 C954    XT     CMP  #'T'              ; Is it a valid 'T' statement?

****** AIM PILOT - 2R1M1A ******

03FA F002           BEQ  TE                ; Branch, it is
03FC 88             DEY                    ; Else back up to original start
03FD 88             DEY
03FE 200704  TE     JSR  PRT               ; Now print the line
0401 205604  ILNEXT JSR  SKPJNK            ; CURAD is set--skip over leading junk
0404 4C1402         JMP  LSTART            ; And go start on the line
0407                .
0407                .      NEXT EOL AND THEN SET UP FOR NEXT LINE
0407                .
0407 B1B7    PRT    LDA  (CURAD),Y         ; Get the current character
0409 C90D           CMP  #CR               ; Do we have it all?
040B F030           BEQ  LINEND            ; Branch if to end of line
040D C924           CMP  #'$'              ; Is it a special one ('$')
040F D024           BNE  CHROUT            ; Branch if not
0411 C8             INY                    ; Also bump to next one
0412 B1D7           LDA  (CURAD),Y         ; Get variable
0414 C93F           CMP  #'?'              ; Is it request for NAME ('?')?
0416 F00F           BEQ  NAMEO             ; Branch if yes
0418 209004         JSR  CNVDSP            ; Convert variable to display
041B A204           LDX  #$04              ; 5 bytes possible
041D                .
041D B5B0    VBDISP LDA  NUMDSP,X          ; Get a character
041F F017           BEQ  CHROUT+3          ; Branch if to end of variable
0421 20BCE9         JSR  OUTALL            ; Also output it
0424 CA             DEX                    ; And count it
0425 10F6           BPL  VBDISP            ; Unconditional loop
0427                .
0427 A23B    NAMEO  LDX  #$3B              ; Remember--it came in backwards
0429 B547           LDA  NAME,X
042B C90D           CMP  #CR               ; Look for end of NAME
042D F009           BEQ  CHROUT+3          ; Branch if to end of NAME
042F 20BCE9         JSR  OUTALL
0432 CA             DEX
0433 10F4           BPL  NAMEO+2           ; Unconditional hop back
0435                .
0435 20BCE9  CHROUT JSR  OUTALL
0438 C8             INY
0439 10CC           BPL  PRT               ; Loop if not too many
043B 302A           BMI  SETBGN            ; Reset to beginning if past the end
043D 20F0E9  LINEND JSR  CRLF              ; Output a CR and the Line Feed
0440                .
0440                .      ENTER HERE TO SKIP A LINE WITHOUT PRINT
0440                .      AND INITIALIZE FOR THE NEXT LINE
0440                .
0440 B1D7    FWD1   LDA  (CURAD),Y         ; Get a character
0442 C90D           CMP  #CR
```

```
0444 F005            BEQ  SCURAD              . Branch if end of line
0446 C8              INY                      . Else bump to next one
0447 10F7            BPL  FWDI                . Loop if not too many
0449 301C            BMI  SETBGN              . Reset to beginning if past the end
044B          ;
044B          .      HERE FIXES UP CURAD TO POINT TO BEGINNING OF A LINE
044B          .      CURAD SHOULD INDEX END OF LINE (WITH Y) ON ENTRY
044B          ;
044D C8       SCURAD INY                      . Bump past the CR
044C 98              TYA                      . Move count to A
044D 18              CLC                      . Clear carry for add
044E 65B7            ADC  CURAD               . Add to low order first
0450 85B7            STA  CURAD               ; And save result
0452 9002            BCC  SKPJNK              . Skip if no carry forward
```

****** AIM PILOT - 2R1M1A ******

```
0454 E6B8            INC  CURAD+1             . Else bump high order
0456          .
0456          .      HERE TO SKIP PAST LEADING JUNK OF A LINE
0456          .
0456 A0FF     SKPJNK LDY  #$FF                . Set up Y this way
0458 C8       SJLOOP INY                      . Increment to next character
0459 B1B7            LDA  (CURAD),Y           . Get character to look at
045B 30FB            BMI  SJLOOP              . Ignore delete character also
045D C92A            CMP  #'*'                . Look for '*' label marker
045F F004            BEQ  SJRTS               . Return if found
0461 C93F            CMP  #'?'                . Look for possible operation character
0463 90F3            BCC  SJLOOP              . Continue skipping if too low
0465 38       SJRTS  SEC                      . Set carry for branches after return
0466 60              RTS                      . Before return
0467          .
0467          . SET UP BEGINNING ADDRESS OF USER AREA
0467          .
0467 20F0E9   SETBGN JSR  CRLF                ; Start on a new line
046A A000            LDY  #$00                . Even page boundary
046C 8400            STY  LST                 . Also set up this guy as default
046E A0EB            LDY  #$EB
0470 84B7            STY  CURAD
0472 A005            LDY  #$05                . User text starts at $0500
0474 8401            STY  LST+1
0476 A004            LDY  #$04
0478 84B8            STY  CURAD+1             . Start of signon message
047A D0DA            BNE  SKPJNK              . Unconditional
047C          .
047C          .      COMPUTE INDEX FOR A VARIABLE
047C          .
047C B1B7     GETIDX LDA  (CURAD),Y           . Get variable letter
047E 38              SEC
047F E941            SBC  #$41                . Subtract 'A' to make relative to zero
0481 0A              ASL  A                   . Times two bytes per variable
0482 AA              TAX                      . Move to index register
0483 60              RTS                      . And return
0484          .
0484          .      TRANSFER A VARIABLE'S DATA TO WORK AREA
0484          .
0484 207C04   VTRANS JSR  GETIDX              . Get index pointer first
0487 B57C            LDA  VARIBS+1,X          . Now move to work area
0489 8542            STA  WORK+1
048B B57B            LDA  VARIBS,X
048D 8541            STA  WORK
048F 60              RTS
0490          .
0490          .      CONVERT A VARIABLE TO DISPLAY FORM
0490          .
0490 208404   CNVDSP JSR  VTRANS              . Move to work area
0493 1017            BPL  ISPLUS              . Branch if positive
0495 A92D            LDA  #'-'                . Else put in minus sign
0497 85B4            STA  NUMDSP+4
0499 F8              SED                      . Set decimal mode indicator
049A 38              SEC
049B A900            LDA  #$00                . Subtract from zero to complement
049D E542            SBC  WORK+1
049F 8542            STA  WORK+1
04A1 A900            LDA  #$00
04A3 E541            SBC  WORK
04A5 8541            STA  WORK
```

****** AIM PILOT - 2R1M1A ******

```
04A7 D8              CLD                      ; Clear decimal mode
04A8 A203            LDX  #$03                . Only 4 positions left
04AA D002            BNE  ISPL1               . Skip index set
04AC          .
04AC A204     ISPLUS LDX  #$04                . Plus has five positions available
04AE 18       ISPL1  CLC
04AF 6646            ROR  SIGNIF
04B1 A541            LDA  WORK                . Get first digit
04B3 20CE04          JSR  TOOUT               . Put to output area
04B6 A542            LDA  WORK+1              . Second digit is high order of this
04B8 4A              LSR  A                   ; Move to low order
04B9 4A              LSR  A
04BA 4A              LSR  A
04BB 4A              LSR  A
04BC 20CE04          JSR  TOOUT
04BF A542            LDA  WORK+1              ; Low order is third digit
04C1 20CE04          JSR  TOOUT
04C4 2446            BIT  SIGNIF              . See if had any significant chars
04C6 3001            BMI  ISPL2               . Skip next if yes
04C8 CA              DEX                      . Else keep the last zero there
04C9 A900     ISPL2  LDA  #$00                . Insert end of line marker
04CB 95B0            STA  NUMDSP,X            .
04CD 60              RTS                      . And return
04CE          ;
04CE          ;      CONVERT CURRENT VALUE TO ASCII AND PUT TO OUTPUT AREA
04CE          .
04CE 290F     TOOUT  AND  #$0F                . Keep only low order
04D0 0930            ORA  #$30                . Make it ASCII
04D2 95B0            STA  NUMDSP,X            . Save regardless
04D4 2446            BIT  SIGNIF              . See if significance started
04D6 3005            BMI  SETSIG              . Yes--all are important now
04D8 C930            CMP  #$30                . Else see if should start  now
04DA D001            BNE  SETSIG              ; Important if not zero
04DC 60              RTS                      . Else return
04DD          .
04DD 38       SETSIG SEC                      . Set significance bit on
04DE 6646            ROR  SIGNIF              . Always
04E0 CA              DEX                      . And point to next available position
04E1 60       PGMEND RTS                      . And then return
04E2          .
04E2          ;
04E2          ; START OF SIGNON MESSAGE
04E2          .
04E2                 * = $04ED
04EB 2A               BYT '*AIM PILOT VER  2R1* ,$0D
04EC 41
04ED 49
04EE 4D
04EF 20
04F0 50
04F1 49
04F2 4C
04F3 4F
04F4 54
04F5 20
04F6 56
04F7 45
04F8 52
04F9 2E
04FA 20
```

****** AIM PILOT - 2R1M1A ******

```
04FB 32
04FC 52
04FD 31
04FE 2A
04FF 0D
0500             583
0500             584             END
END 6502*CROSS.      Time    1739 ms      No  Errors

@RESUME.EU
EOF:631
0.
```

[Explenation of the exemples]

For demonstretion purposes, I heve Included three exemple progrems. The first program Inpute two numbers (one et e time) and puts them Into verlebles A end B, respectively. The two verlebles ere then edded together and placed Into C. The mechlne lenguege routines ere quick 'n' dirty; I.e. you must enter the number es e four-digit string. If you wish to Input negative numbers, they must be Inputted In 10's complement form. Anyone seriously using these routines would do well to write them over.

The second program demonstrates where Tiny Pilot really stends out, which Is In educetionel purposes. After running thie progrem, the ueer hes ell he needs to know to load end save programs on tepe.

The third program should prove quite useful to anyone who wante to perform program loops. It teets variable A to see If It Is equel to zero and sets the metch flag if so.

For people who wish to experiment with the H: command, remember the high order byte of A Is et $7B, low order et $7C. Continue counting up for the locations of other varlebles. The ANSWER fleld sterts et $3E end works Its wey down In memory.

```
R:EXAMPLE #1
R:
R:INPUT TWO NUMBERS,
R:PUT THEM IN A & B,
R:ADD THEM TOGETHER,
R:AND PRINT RESULT.
R:
P:ON
T:FIRST NUMBER=  ?
A:
H:0900
R:TRANSFER TO A
T:
T:SECOND NUMBER=  ?
A:
H:0980
R:TRANSFER TO B
C:C=A+B
T:
T:THE SUM IS $C
P:OFF
S:
<K>*=0900
/19
 0900 48 PHA
 0901 20 JSR EB9E
 0904 A2 LDX #3B
 0906 B5 LDA 03,X
 0908 20 JSR EA7D
 090B CA DEX
 090C B5 LDA 03,X
 090E 20 JSR EA84
 0911 85 STA 7B
 0913 CA DEX
 0914 B5 LDA 03,X
 0916 20 JSR EA7D
 0919 CA DEX
 091A B5 LDA 03,X
 091C 20 JSR EA84
 091F 85 STA 7C
 0921 20 JSR EBAC
 0924 68 PLA
 0925 60 RTS
```

```
<K>*=0980
/19
 0980 48 PHA
 0981 20 JSR EB9E
 0984 A2 LDX #38
 0986 B5 LDA 03,X
 0988 20 JSR EA7D
 098B CA DEX
 098C B5 LDA 03,X
 098E 20 JSR EA84
 0991 85 STA 7D
 0993 CA DEX
 0994 B5 LDA 03,X
 0996 20 JSR EA7D
 0999 CA DEX
 099A B5 LDA 03,X
 099C 20 JSR EA84
 099F 85 STA 7E
 09A1 20 JSR EBAC
 09A4 68 PLA
 09A5 60 RTS
<I>
*AIM PILOT VER. 2R1*
FIRST NUMBER=  ?
?0357
SECOND NUMBER=  ?
?0231
THE SUM IS 588
```

```
R:EXAMPLE #2
R:
R:TEACHING PROGRAM--
R:HOW TO USE THE
R:TINY PILOT.
R:
P:ON
T:
T:
T:THIS PROGRAM WILL
T:TEACH YOU HOW TO
T:LOAD AND USE TINY
T:PILOT PROGRAMS.
T:WHAT'S YOUR NAME?
T:
P:OFF
?:
P:ON
T:OKAY, $?,
T:THE FIRST ITEM OF
T:BUSINESS IS TO
T:LEARN HOW TO LOAD
T:UP THE INTERPRETER
T:INTO MEMORY.
T:
T:DO YOU KNOW HOW TO
T:DO THIS, $? ?
T:
U:B
YJ:L
T:FIRST, MAKE SURE
T:THE CONNECTOR IS
T:HOOKED TO THE TAPE
T:DRIVE AND THE COM-
T:PUTER RIGHT. THEN,
T:TYPE "L" IF YOU
T:ARE IN THE MONITOR
T:OR THE ESCAPE IF
T:YOU'RE SOMEWHERE
T:ELSE, THEN TYPE
T:"L." THE DISPLAY
T:WILL SHOW 'IN='
T:TYPE "T" FOR TAPE,
T:THEN THE DISPLAY
T:WILL SHOW 'F=' FOR
T:THE FILE NAME. OF
T:COURSE, YOU SHOULD
T:TYPE "PILOT". THEN
T:THE COMPUTER WILL
T:ASK FOR THE TAPE
T:DRIVE NUMBER OF
T:THE TAPE YOU WANT.
T:TYPE "1", PUSH THE
T:PLAY BUTTON ON THE
T:TAPE DRIVE, AND
T:HIT RETURN. MAKE
T:SURE THE TAPE IS
T:NOT PAST THE START
T:OF PILOT.
T:THE DISPLAY WILL
T:TELL YOU WHEN IT
T:HAS FOUND PILOT.
T:WHEN THE DISPLAY
T:IS CLEAR, YOU CAN
T:START THE EDITOR
T:AT LOCATION 0500
T:AND TYPE IN OR
T:LOAD IN YOUR TEXT
*LT:DO YOU KNOW HOW
T:TO GET TEXT FROM
T:THE TAPE, $? ?
U:B
```

```
YJ:E
T:TO LOAD TEXT FROM
T:TAPE INTO THE AIM
T:EDITOR, TYPE "R".
T:THE RESPONSES WILL
T:BE THE SAME AS
T:BEFORE. YOU SHOULD
T:ANSWER WITH THE
T:APPROPRIATE RE-
T:SPONSES. YOU CAN
T:THEN MAKE CHANGES
T:TO THE PROGRAM, AS
T:YOU WILL STILL BE
T:IN THE EDITOR.
T:
*ET:WHEN A PROGRAM
T:IS RUNNING RIGHT,
T:YOU CAN SAVE IT ON
T:TAPE. DO YOU KNOW
T:HOW TO DO THIS?
U:B
YJ:Z
T:TO SAVE YOUR
T:PILOT PROGRAM ON
T:TAPE, MAKE SURE
T:THAT YOU ARE IN
T:THE EDITOR. THEN
T:TYPE "L". AIM WILL
T:DISPLAY 'OUT='.
T:YOU TYPE "T". THEN
T:THE DISPLAY WILL
T:PROMPT FOR A FILE
T:NAME. GIVE IT A
T:NAME OF 5 OR LESS
T:CHARACTERS AND HIT
T:RETURN. AIM WILL
T:THEN PROMPT FOR
T:WHICH TAPE DRIVE
T:YOU ARE USING.
T:GIVE IT THE NUMBER
T:OF THE TAPE DRIVE
T:YOU ARE USING.
*ZT:
T:WELL, $?,
T:THAT'S ALL YOU
T:NEED TO KNOW TO
T:USE TINY PILOT
T:PROGRAMS. GOOD
T:LUCK!
T:
T:
P:OFF
S:
*BP:OFF
A:
```

```
M:Y
P:ON
T:
T:
E:

R:EXAMPLE #3
R:
R:THIS PROGRAM WILL
R:DEMONSTRATE HOW
R:TO SET UP A LOOP
R:BY USING THE H:
R:COMMAND TO SET THE
R:MATCH FLAG.
R:
C:A=10
P:ON
T:COUNTDOWN....
*LT:$A
C:A=A-1
H:0900
R:TEST FOR ZERO AND
R:SET MATCH FLAG IF
R:RESULT IS ZERO.
NJ:L
T:DONE!!
S:
<K>*=0900
/11
 0900 48 PHA
 0901 A5 LDA 7B
 0903 D0 BNE 090F
 0905 A5 LDA 7C
 0907 D0 BNE 090F
 0909 A9 LDA #59
 090B 85 STA 02
 090D 68 PLA
 090E 60 RTS
 090F A9 LDA #4E
 0911 4C JMP 0905
 <E>

*AIM PILOT VER. 2R1*
COUNTDOWN....
10
9
8
7
6
5
4
3
2
1
DONE!!
```

MEAN 14: A Pseudo-Machine Floating Point Processor for the APPLE II

Modelled efter the Sweet 16, this progrem supports e lerge veriety of methematicel operetions on five-byte floeting point values. This 'processor' cen greetly simplify and enhence your mathemeticel processing power.

R.M. Mottola
Cyborg Corp.
342 Western Ave.
Boston, MA 02135

In the beginning of the life of the Apple II computer, and obstacle had to be overcome in the writing of the firmware. As we know, the 6502 is an eight bit microprocessor, but all too frequently routines require numeric operations involving double precision integers. Repeating common operations every time they are required could be done, but it is not very space efficient. For that matter, performing the requisite register set-ups to use some general purpose subroutines can also deplete available memory space, if the routines are called frequently. What was needed was an arithmetic processor that could handle two-byte integers. So, pseudo-machine processor, which in reality, is a machine language program that behaves like a processor.

This elegant solution is called the "SWEET 16 PSEUDO-MACHINE INTERPRETER" and is known and used by many Appfe programmers. ft lives from $F689 to F7FA on the FO Integer Basic ROM found in regular Apple II computers. From a software point of view. It is used very much like you woufd use a Microprocessor. Programming it requires various instructions and operands. Hand assembly is easy because the instruction set isn't long and the format of the operators is very straight-forward. A popular resident asembler, the Lisa assembler by Randall Hyde, will even assemble Sweet 16 mnemonics.

The Mean 14 pseudo-machine floating point processor was modefled after the Sweet 16. It too is programmed like a hardware processor. Instead of being designed to process two-byte integers, the Mean 14 can perform many mathematical operations on five-byte floating point values. These values are formatted in the standard Applesoft variabfe representation described in the Applesoft manual.

What It Is Used For

The Mean 14 processor was written to facilitate floating point machine language programming on an Apple fl Plus or a standard Apple II with Applesoft ROM card. Since Apple does not provide any documentation for the floating point routines in Applesoft, it is pretty difficult for those wishing to write floating point routines in assembly language. Even knowing the locations and entry requirements of those routines is only partially helpful if either complex or repetitive functions must be performed. Of course, you could always write your more Involved functions in Applesoft Basic, but the Mean 14 will afways perform at least ten times as fast and probably much more. The reason for this is simply that the Mean 14 has little of the interpreter overhead that Appfesoft has. Using the example of adding two values, If Applesoft is used, and the values are represented as variables which have not been used before, Applesoft must allocate space for them first. And If arrays have been dimensioned. They must be moved up to make space for the new variables. ff the variables or arrays happen to collide with strings, then string "house-cleaning" must take place. In machine terms, alf this takes an awful lot of time. As an added kicker, even more time must be allowed if you use constants instead of variables.

On the other hand, Mean 14 doesn't have to do all of this. Its fnterpreter overhead is very small and since you, the programmer, supply the operand either by specifying pointers or, in the Immediate Mode, by actually supplying the floating point value, the floating point routines don't have to search for or convert anything. Mean 14 spends its time processing numbers — not trying to find them or converting ASCII strings fnto them.

What ft Does

Mean 14 Is a very simple kind of interpreter. You give it a number and it looks it up in a tabfe where it picks up the address of the subroutine which performs the specific function required. Most of those functions already exist in Applesoft. Some require set-ups to make entry and exit easier. In all cases, the instruction set has been designed to make straight line machine language floating point arithemetic a lot easier.

That last line indicates one of the possible shortcomings of the Mean 14 for your particular floating point requirement. It can process data only in a strafght line. At present, it contains no conditionals fn the instruction set. This apparent problem

isn't really all that bad when you actually use the Mean 14. For my own applications, I've found that testing, branching, and loop operations can best be handled outside of Mean 14, in 6502 assembly language. This is because, relative to the amount of time it takes even the simplest floating point operation to execute, all sorts of branching and testing, including entries and exits into and out of Mean 14, can be accomplished very quickly. For this reason, conditionals were left out of the Mean 14's instruction set. But that certainly doesn't mean that you couldn't add them if you particular application required them.

Using Mean 14

Making use of the Mean 14 processor in you machine language programs is easy. The only prerequisite, besides a working knowledge of assembly language, is a fundamental knowledge of the format of Applesoft variables. For more on this, including a handy utility program that converts any value to its floating point equivalent, see the predecessor to the article, "Applesoft Floating Point Routines, MICRO 27:53". Once this is understood, Mean 14 assembly is very straight-forward.

1. Note that Mean 14 and the Applesoft subroutines that it calls could leave any and all registers in an undeterminable state. If you need certain registers in specific states, its a good idea to write your self both a Save and a Restore routine and remember to JSR to the Save before entering Mean 14. You could even add these routines to the Mean 14 entry and exits if you like.

2. Enter Mean 14 with a JSR to MEAN14 ($8E00 in the source listing provided.) All code between this JSR and a Mean 14 "RET" will be interpeted by the Mean 14 processor. Remember that byte sequence is a function of the addressing mode. In the Implied mode, any operator is followed by the next operator. In Immediate mode, an operator is immediately followed by a five byte operand (constant) in Applesoft floating point variable format. In the Absolute mode, the operator must be followed by a two byte pointer to the first memory location containing a floating point value. In the Indirect mode, the operator is followed by a pointer which points to a pointer which points to a floating point value. Remember, all pointers must be in standard 6502 low-byte, high-byte order.

3. Consider the following section of code:

```
2000  SUB1     STY YSAVE         ; SAVE Y
2002           STX XSAVE         ; SAVE X
2004           JSR MEAN14        ; ENTER MEAN 14
2007           DFB C0 00 03      ; *LDA $300
200A           DFB C4 05 03      ; *ADD $305
200D           DFB 45 81 00
2010           DFB 00 00 00      ; *SUB #1
2013           DFB 0C            ; *ABS
2014           DFB 81 40 03      ; *STA ($340)
2017           DFB 11            ; *RET
2018           LDX XSAVE         ; RESTORE X
201A           LDY YSAVE         ; RESTORE Y
201C           RTS
```

Both the X and Y registers were saved before entering Mean 14 in this example. To make the code representation less confusing, its a good idea to show the Mean 14 mnemonic equivalents of the defined bytes in the comments field. I like to designate them with an asterisk but any appropriate scheme should do.

4. If your machine language routines are to be called from Basic and if values obtained from Mean 14 operations will be used by Basic, you might want to store values directly into the memory locations allocated to Applesoft variables. This will make the results of your machine language calculations directly available to Basic. Although there are subroutines in Applesoft to fine a variable by its name, they can take a lot of time to execute. An easier approach is to "know" where your variables are by allocating them first, in your Basic program. Thus, if the first line of your program is:

10 A = 0:B = 0:C = 0:D = 0

then you'll know that the first variable is A,the second is B, etc. The pointer at locations $69,$69A tells you the beginning of the simple variable space, so you should be all set.

5. Be careful to avoid floating point errors such as Overflow and Division by Zero, as Applesoft routines tend to dump you into Basic if an error occurs. A scheme to avoid this has also been outlined in "Applesoft Floatng Point Subroutines".

6. Good Luck!

μ

Format Of Mean 14 Operators

Mean 14 Instructions are represented as single byte numberic values. Two quantities are represented in this byte — instruction and addressing mode. Since there was room to spare (there are only four addressing modes and twenty some-odd instructions) a very simple scheme was devised to include both. There are also many unused values so the instruction set could easily be expanded. An instruction is represented with the two high order bits indicating the adressing mode and the lower six bits indicating the operation

7	6	5	4	3	2	1	0
Addressing Mode		Instruction					

Mean 14 Addressing Modes

The Mean 14 pseudo-machine processor instructions use four different addressing modes. They are:

IMMEDIATE
ABSOLUTE
INDIRECT
IMPLIED

IMMEDIATE· Just like any processor, the Mean 14 instructions

that allow immediate addressing use the value following an operator in memory for the operand. Since we deal with floating point values, the five memory locations following the operator must contain the floating point operand. this must be in Applesoft variable format.

EX. Load FPAC1 with the value "0"

```
40        00 00 00 00 00     LDA#0
 \               \             \
OPERATOR       OPERAND        SYM-
BOLIC
```

ABSOLUTE- The two bytes that follow the instruction (operator) in the absolute mode must contain the address of the first byte of the desired buffer.

EX. Store FPAC1 in locations $1F00-$1F04

```
C1          00 1F    STA $1F00-$1F04
 \            \                 \
OPERATOR   OPERAND            SYM-
BOLIC
```

INDIRECT- In this addressing mode, the two bytes that follow the operator must contain the address of a two byte pointer which points to the first byte of the buffer. This addressing mode is useful when loop processing an number of variables. It allows the pointer to the variable to be changed and, since the pointer is not a part of the Mean 14 object code, you needn't write self modifying code to perform a loop. Again, both the operand and the pointer must be represented in the low byte, high byte format.

EX. Store FPAC1 in $2FF0-$2FF4

81 00 20   STA($2000)

Where $2000,$2001 point at $2FF0

IMPLIED- Certain Instructions perform operations which do not involve variables. There include register functions and exits form Mean 14.

EX. Transfer FPAC1 to FPAC2
02   TAB
EX. Exit Mean 14
11 RET

MEAN 14 INSTRUCTION SET

```
LDA   Load FPAC1 with memory               M --> FPAC1

        IMMEDIATE = $40
        ABSOLUTE  = $C0
        INDIRECT  = $80
------------------------------------------------------------
STA   Store FPAC1 in memory                FPAC1 --> M

        ABSOLUTE  = $C1
        INDIRECT  = $81
------------------------------------------------------------
TAD   Transfer FPAC1 to FPAC2         FPAC1 --> FPAC2

        IMPLIED   = $02
------------------------------------------------------------
TBA   Transfer FPAC2 to FPAC1         FPAC2 --> FPAC1

        IMPLIED   = $03
------------------------------------------------------------
ADD   Add memory to FPAC1             M + FPAC1 --> FPAC1

        IMMEDIATE = $44
        ABSOLUTE  = $C4
        INDIRECT  = $84
------------------------------------------------------------
SUB   Subtract FPAC1 from memory    M - FPAC1 --> FPAC1

        IMMEDIATE = $45
        ABSOLUTE  = $C5
        INDIRECT  = $85
------------------------------------------------------------
MUL   Memory times FPAC1              M * FPACI --> FPAC1

        IMMEDIATE = $46
        ADSOLUTE  = $C6
        INDIRECT  = $86
------------------------------------------------------------
DIV   Memory divided by FPACI         M / FPAC1 --> FPAC1

        IMMEDIATE = $47
        ABSOLUTE  = $C7
        INDIRECT  = $87
------------------------------------------------------------
NOP   No operation                                MPC + 1

        IMPLIED   = $08
------------------------------------------------------------
SQR   Square root of FPAC1             √FPAC1 --> FPACI

        IMPLIED   = $09
------------------------------------------------------------
EXP    FPAC2 raised to the power     FPAC2 ^ M --> FPAC1
        of memory

        IMMEDIATE = $4A
        ADSOLUTE  = $CA
        INDIRECT  = $8A
```

```
INT    Integer value of FPAC1        INT ( FPAC1 ) --> FPAC1

         IMPLIED    = $0B

----------------------------------------------------------------

ABS    Absolute value of FPAC1       ABS ( FPAC1 ) --> FPAC1

         IMPLIED    = $0C

----------------------------------------------------------------

SGN    Value of the sign of          SGN ( FPAC1 ) --> FPAC1
        FPAC1

         IMPLIED    = $0D

----------------------------------------------------------------

LOG    Natural log of FPAC1          LOG ( FPAC1 ) --> FPAC1

         IMPLIED    = $0E

----------------------------------------------------------------

CVA    Convert two-byte integer                 M% --> FPAC1
          in Applesoft integer variable
          format to its floating point
          equivalent.

         ABSOLUTE   = $CF
         INDIRECT   = $8F

----------------------------------------------------------------

CVB    Convert two-byte integer              ML, MH --> FPAC1
          in 6502 format to its floating
          point equivalent.

         ABSOLUTE   = $D0
         INDIRECT   = $90

----------------------------------------------------------------

RET    Exit MEAN 14                            MPC --> PC

         IMPLIED    = $11

----------------------------------------------------------------
```



```
**END OF PASS 1
**END OF PASS 2

0800                ;**********************
0800                ;*                    *
0800                ;*      MEAN-14       *
0800                ;*   PSEUDO-MACHINE   *
0800                ;*   FLOATING POINT   *
0800                ;*   PROCESSOR V1.0   *
0800                ;*                    *
0800                ;*   R M. MOTTOLA     *
0800                ;*   10/79            *
0800                ;*                    *
0800                ;**********************
0800                ;
0800                ;
0800                ;SOFTWARE ADDRESSES
0800                ;
0800                TEMPL   EPZ $1E
0800                TEMPH   EPZ $1F
0800                MPCL    EPZ $4C
0800                MPCH    EPZ $4D
0800                FPACI   EPZ $9D
0800                FPAC2   EPZ $A5
0800                ;
0800                ;FIRMWARE ADDRESSES
0800                ;
0800                INT>FP  EQU $E2F2
0800                FPSUB   EQU $E7A7
0800                FPADD   EQU $E7BE
0800                FPLOG   EQU $E941
0800                FPMUL   EQU $E97F
0800                FPDIV1  EQU $EA66
0800                FPLOAD  EQU $EAF9
0800                FPSTR   EQU $EB2B
0800                TR2>1   EQU $EB53
0800                TRI>2   EQU $EB63
0800                FPSGN   EQU $EB90
0800                FPABS   EQU $EBAF
0800                FPINT   EQU $EC23
0800                FPSQR   EQU $EE8D
0800                FPEXP   EQU $EE94
0800                ;
8E00                        ORG $8E00
8E00                        OBJ $800
8E00                ;
8E00                ;MEAN 14 PSEUDO-MACHINE
8E00                ;FLOATING POINT PROCESSOR
8E00                ;
8E00 68             MEAN14  PLA                 ,GET M14 CODE LOCATION
8E01 854C                   STA MPCL            ;FROM RETURN ADDRESS
8E03 68                     PLA
8E04 854D                   STA MPCH
8E06 205F8E                 JSR PCINC
8E09 200F8E         M14A    JSR M14B
8E0C 4C098E                 JMP M14A
8E0F A000           MI4B    LDY #$0
8EII B14C                   LDA (MPCL).Y        ;GET ONE INSTRUCTION
8E13 AA                     TAX
8E14 293F                   AND #$3F            ;GET CORRECT SUBROUTINE
8E16 0A                     ASL                 ;ADDRESS FROM TABLE
```

```
8E17 A8                TAY
8E18 C8                INY
8E19 B9A08E            LDA SUBTBL,Y  ;AND SHOVE 1T
8E1C 48                PHA
8E1D 88                DEY
8E1E B9A08E            LDA SUBTBL,Y
8E21 48                PHA
8E22 205F8E            JSR PCINC     ;1NCREM. M14 P.C. COUNT
8E25 8A                TXA
8E26 29C0              AND #$C0      ;GET ADDRESSING MODE
8E28 F034              BEQ M14G      ;1MPL1ED?
8E2A 1020              BPL M14D      ;IMMEDIATE?
8E2C 2940              AND #$40
8E2E D013              BNE M14C      ;ABSOLUTE?
8E3D B14C              LDA (MPCL),Y  ;INDIRECT
8E32 851E              STA TEMPL     ;GET POINTER TO ADDRESS
8E34 C8                INY           ;OF OPERAND
8E35 B14C              LDA (MPCL),Y
8E37 851F              STA TEMPH
8E39 88                DEY
8E3A B11E              LDA (TEMPL),Y
8E3C 48                PHA
8E3D C8                INY
8E3E D11E              LDA (TEMPL),Y
8E40 48                PHA
8E41 9013              BCC M14E
8E43 B14C       M14C   LDA (MPCL),Y  ;GET ADDRESS OF
8E45 48                PHA           ;OPERAND
8E46 C8                INY
8E47 B14C              LDA (MPCL),Y
8E49 48                PHA
8E4A 900A              BCC M14E
8E4C A54C       M14D   LDA MPCL      ;SAVE P.C. AS ADDRESS
8E4E 48                PHA           ;OF IMMEDIATE OPERAND
8E4F A54D              LDA MPCH
8E51 48                PHA
8E52 A905              LDA #$5       ;AND OFFSET P.C. 5 BYTES
8E54 9002              BCC M14F
8E56 A902       M14E   LDA #$2       ;OFFSET P.C. 2 BYTES
8E58 20618E     M14F   JSR PCADD
8E5B 68                PLA           ;PULL OPERAND ADDRESS
                                            AND TRANSFER
8E5C A8                TAY           ;TO A AND Y REGS FOR SUBS
8E5D 68                PLA
8E5E 60         M14G   RTS           ;JMP VIA RTS
8E5F            ;
8E5F A901       PC1NC  LDA #$1
8E61 18         PCADD  CLC
8E62 654C              ADC MPCL
8E64 854C              STA MPCL
8E66 9003              BCC PC1
8E68 E64D              1NC MPCH
8E6A 18                CLC
8E6B A000       PC1    LDY #$0
8E6D 60                RTS
8E6E            ;
8E6E AA         STR    TAX
8E6F 4C2BEB            JMP FPSTR
8E72 851E       CONV1  STA TEMPL
8E74 841F              STY TEMPH
8E76 A000              LDY #$0
8E78 B11E              LDA (TEMPL),Y
8E7A 48                PHA
8E7B C8                INY
8E7C B11E       C1A    LDA (TEMPL),Y
8E7E A8                TAY
8E7F 68                PLA
8E80 20F2E2            JSR INT>FP
8E83 A5A2              LDA FPAC1+$5
8E85 1007              BPL NOOP
8E87 A9C4              LDA #VALUE1
8E89 A08E              LDY /VALUE1
8E8B 20BEE7            JSR FPADD
8E8E 60         NOOP   RTS
8E8F 851E       CONV2  STA TEMPL
8E91 841F              STY TEMPH
8E93 A001              LDY #$1
8E95 B11E              LDA (TEMPL),Y
8E97 48                PHA
8E98 88                DEY
8E99 F0E1              BEQ C1A
8E9B 68         RETURN PLA           ;PULL MEAN 14 RETURN
8E9C 68                PLA           ;ADDRESS FROM STACK
8E9D 6C4C00            JMP (MPCL)
```

```
8EA0            ;
8EA0            ;
8EA0            ;SUBROUTINE ADDRESS TABLE
8EA0            ;
8EA0 F8EA       SUBTBL ADR FPLDAD-$1
8EA2 6D8E              ADR STR-$1
8EA4 62EB              ADR TR1>2-$1
8EA6 52E8              ADR TR2>1-$1
8EA8 BDE7              ADR FPADD-$1
8EAA A6E7              ADR FPSUB-$1
8EAC 7EE9              ADR FPMUL-$1
8EAE 65EA              ADR FPD1V1-$1
8EB0 8D8E              ADR NOOP-$1
8EB2 8CEE              ADR FPSQR-$1
8EB4 93EE              ADR FPEXP-$1
8EB6 22EC              ADR FP1NT-$1
8EB8 AEE8              ADR FPABS-$1
8EBA 8FEB              ADR FPSGN-$1
8EBC 40E9              ADR FPLOG-$1
8EBE 718E              ADR CONV1-$1
8EC0 8E8E              ADR CDNV2-$1
8EC2 9A8E              ADR RETURN-$1
8EC4            ;
8EC4            ;FLOAT1NG POINT CONSTANTS
8EC4            ;
8EC4 910000     VALUE1 HEX 9100000000      ; % 65536
8EC7 0000
8EC9            ;
8EC9            ;
8EC9            ;
8EC9            LENGTH EQU *-MEAN14
               END    END
```

```
************************
*                      *
* SYMBOL TABLE -- V 1.5 *
*                      *
************************
```

LABEL. LOC. LABEL. LOC. LABEL. LOC.

** ZERD PAGE VARIABLES:

TEMPL	001E	TEMPH	001F	MPCL	004C
MPCH	004D	FPAC1	009D	FPAC2	00A5

TEMPL	001E	TEMPH	001F	MPCL	004C
MPCH	004D	FPAC1	009D	FPAC2	00A5
INT>FP	E2F2	FPSUB	E7A7	FPADD	E7BE
FPLOG	E941	FPMUL	E97F	FPDIV1	EA66
FPLOAD	EAF9	FPSTR	EB2B	TR2>1	EB53
TR1>2	EB63	FPSGN	EB90	FPABS	EBAF
FPINT	EC23	FPSQR	EE8D	FPEXP	EE94
MEAN14	8E00	M14A	8E09	M14B	8E0F
M14C	8E43	M14D	8E4C	M14E	8E56
M14F	8E58	M14G	8E5E	PCINC	8E5F
PCADD	8E61	PC1	8E6B	STR	8E6E
CONV1	8E72	C1A	8E7C	NOOP	8E8E
CONV2	8E8F	RETURN	8E9B	SUBTBL	8EA0
VALUE1	8EC4	END	8EC9		

The MICRO Software Catalog: XXIV

Software announcements for the 6502 based systems

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **Spaad Raading and Comprahension**
System: PET/CBM
Memory: 16 or 32K
Hardware: cassette drive
Description: A ftexible and comprehensive system in which the teacher creates a permanent test and question data file on a cassette. This file is used by one ot the other programs to give a rapid scan, and then a timed read scan, followed by questions which are corrected. All statistics including reading speed, in words per minute, are then printed on the screen (printer optional). The system has many options including: adjustable read rate, various methods of displaying the text for reading, and directions tor customizing the programs for individual perterences and teaching strategies.
Copies: Just released
Price: $49.95 (extra manuals, $2.00)
Includes: Six programs, sampte data file, manual, all in a six cassette plastic binder
Author: **Richard A. Brown, Ph.D.**
Available: **Abbott Educational Software**
334 Westwood Avenue
E. Longmeadow, MA 01028

Name: **WP-INT**
System: Ohio Scientific
Memory: 48K RAM
Language: Basic, 6502 Assembler
Hardware: C2-OEM and C3 series
Description: A form letter generation package that unites two OSI sottware systems, WP-2 and OS-DMS. The system extracts information from OS-DMS data files to prepare from letters with OSI's word processor, WP-2. Supplied on two floppy disks.
Price: $80.00
does not Include OS-DMS or WP-2
Available: **DCS Software Products**
2729 Lowery Court
Zion, IL 60099

Name: **Copy T-Fite**
System: Apple ii, Appte II plus
Memory: 16K with ROM
32K without
Language: Applesoft
Hardware: Disk II
Description: Copies any EXEC file or sequential TEXT tile to another disk. You can display the files tield by tield and directly change any field in the TEXT file before copying. Modify your own EXEC programs directly without going thru the 'Make-EXEC' routine. Lets you display and study professional EXEC programs. Self-prompting. Simpte and easy to use.
Price: $15.95
Includes: Disk with program and instructions
Author: **David Waston**
Available: **David Weston**
P.O.Box 25943
Los Angeles, CA 90025

Name: **Supersort**
System: PET/CBM computers
Memory: 851 bytes at the top of memory, plus parts of the second cassette bufter. The demo program uses 7k.
Language: Machine, the loader and demonstrator programs are In Basic.
Description: Enchanced version of KEYSORT (MICRO 23 & 24). It shares KEYSORT's advantages, and adds several features requested by MICRO readers: Sorts 1 or 2 dimesion arrays of strings or Integers on any of up to 127 tietds, with optionat subsorting on macth to any other filed or tields, all in ascending or descending order. Detimiters are not needed with this, and data may be easlly viewed without using MtD$ functions needed by KEYSORT.
Copies: Just released
Price: $34.95
Includes: tull instructions, complete demo program, assembly source listing
Author: **James Strasma**
Avatlabte: **Programma International**
3400 Wilshire Blvd.
Los Angeles, CA 90010

Name: **Video Massaga Display**
System: Appte II
Memory: 48K RAM
Language: Apple Integer Basic
Hardware: Color tv set, RF Modulator or cotor Video Monitor, Mountain Hardware Clock, Apple Disk Drive
Description: Converts a computer into an etectronic bulletin board. A set of slmple commands allows the user to define a serles of "slides" that can be displayed in any sequence and for varying amounts of time. Low resolution dislplays otter normal-sized characters in normal,

reverse, or blinking video. High resolution Displays permit intermixed characters of three different sizes in either normal or reverse video. In addition, the background of the "slide" can be displayed in any high resolution color. Professional version, model VMP, is available for the Apple II. Provides hardcopy slide logs for use by television stations.

Price: VMD—$149.00
VMP—$199.00

Available: **Serendipity Systems, Inc.**
225 Elmira Road
Ithaca, N.Y. 14850

Name: **Micro-Inventory (MIN)**
System: Apple II, Applesoft Firmware Board
Memory: 48K RAM
Hardware: Tv set, RF modulator or video Monitor, Apple disk drive, Optional printer

Description: Developed with the particular needs of small businesses in mind, this package provides owners of such firms with effective inventory control. Each inventory item is assigned a unique Item Identifier by the user, and data is stored in logical files. Although the capacity of the system is normally limited to six files of 200 items each, multiple diskette drives can be used to accommodate additional inventory items. Reports provided include Items On File, Items On Hand, Items On Order, etc. Each report can be generated to include all inventory items or only those specified by the user.

Price: $149.00
Available: **Serendiptiy Systems, Inc.**
225 Elmira Road
Ithaca, N.Y. 14850

Name: **Micro-General Ledger**
System: Apple II Plus, Apple II w/Applesoft Firmware Board
Memory: 48K RAM
Language: Apple Integer BASIC
Hardware: Tv set with RF modulator or video monitor, Apple disk drive

Description: Designed with the needs of very small businesses in mind, MGL allows the user to retain financial control while requiring a minimum knowledge of accounting. It features a user-defined chart of accounts, interactive data entry and editing routines, extensive error detection devices, and automatic end-of-month and end-of-year resetting of totals. Reports produced in Sheet, and an Accounts Reconcilla tion Report. The system can accommodate 75 accounts and each account may be assigned a total of nine sub-account numbers.

Price: $149.00
Available: **Serendipity Systems, Inc.**
225 Elmira Road
Ithaca, N.Y. 14850

Name: **AMS/OIL Inventory/Sales/Price List**
System: Apple II
Memory: 32K
Language: ROM Applesoft
Hardware: Disk II

Description: Program maintains price list, handles sales both retail and wholesale, with or without shipping, maintains inventory with monthly and year-to-date formats. Creates, sorts and provides easy update to price lists. Can be used for AMWAY as well.

Price: $30.00 includes disk
$15.00 w/o inventory program
Author: **Allan Blackburn**
Available: **AWB's**
1226 Wade Hampton
Fort Worth, TX 76126

Name: **Satellite**
System: Apple II, Apple II plus
Memory: 32K
Language: Applesoft ROM/RAM

Description: Provides the amateur radio operator or shortwave listener with all data necessary to track spacecraft in either circular or elliptical orbits. It will provide enough information so the operator can aim an antenna at the spacecraft and keep up with it as it crosses the sky. The program has two main modes. Information for the satellites is provided in a number of publications, includeing OST, Worldradio, and '73 magazines. Program to screen or printer.

Copies: Just released
Price: $14.95 cassette, or user provided diskette
$19.95 on diskette by author, postpaid. Specify Applesoft RAM or ROM
Author: **Al Jensen** WA7TIB
Available: **Al Jensen**
19111 First Avenue NW
Seattle, WA 98177

Name: **The Voice**
System: Apple II, Apple II plus
Memory: 48K
Hardware: No special

Description: Gives your apple the power of speech! Use the standard voice vocabulary to speak an endless combination of phrases and sentences, or easily record your own vocabulary set to make your Apple say anything you like. Each data disk can store up to 80 words or phrases which can be sorted for quick reference. What's more, the Voice allows you to speak from any Basic program by using Print Commands. Guaranteed to be the best Apple voice program available at any price.

Price: $39.95 disk
Available: **Muse Software**
330 N. Charles Street
Baltimore, MD 21201

Name: **Elementary Math Edu-Disk**
System: Apple II
Memory: 48K
Language: Integer Basic

Description: Written and designed by a professional educator. Contains an arithmetic readiness test and four interactive lessons designed to teach elementary math skills on nine different skill levels. This program is self-demonstrating and requires little or no instructor assistance. Recommended for the student with no prior arithmetic experience, and as a supplement in higher level remedial situations.

Price: $39.95 disk
Available: **Muse Software**
330 N. Charles Street
Baltimore, MD 21201

Name: **Inventory Program**
System: Apple II, Apple II Plus
Memory: 48K (Firmware card on Apple II)
Language: Applesott, Assembly
Hardware: 2 Disk drives, 132 column printer

Description: Maintains a complete inventory on up to 800 items. Every category included to back order as well as LOC, COST, etc. Generates search reports, keeps running account of what was sold YTD and much more.

Price: $140.00 with manual
Author: **Gary E. Haffer**
Available: **Software Technology for Computers**
P.O.Box 428
Belmont, MA 02178

6502 Bibliography: Part XXIV

Continuing bibliography of 6502 related material

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

687 (cont'd.) COMPUTE 3, (Mar./Apr., 1980)

Butterfield, Jim, "Memo to Machine Language Programmers," pg. 96.
Ways to find zero page space on the New PET ROM.

Rehnke, Eric, "Read PET Tapes with your AIM," pg. 102 - 104.
Use the General Instruments AY3-8910 device to generate music on 6502 boards.

Zumchack, Gene, "Nuts and Volts," pg. 105-107.
READ/WRITE timing on the 6502

Rehnke, Eric, "Read PET Tapes with Your AIM," Pg. 110 - 112
This program opens up PET software to the AIM owners.

Herman, Harvey B., "KIMEX — 1," pg. 113
PROM, RAM and I/O expansion for the KIM

Carlson, Edward H., "Fast Tape Read/Write Programs for your OSI," Pg 115-117
Add this useful utility to your OSI C1 or C2 machines.

Flacco, Roy, "Applications Review: Logic Analyzer for KIM," Pg. 118 - 120
A review of a useful piece of hardware.

688. Softside 1, No. 3 (March 1980)

Porter, Gale, "The Care and Feeding of Integer Hi-Res," pg. 9 -11.
Pep up your Apple Integer Programs with Hires Graphics.

Anon, "Programming Tips," pg. 15-16.
A routine for rounding off decimal numbers, on the Apple.

Chipchase, Frank D., "Renumber and Merge the Easy Way," pg. 19-21.
Make this useful utility into an Exec program, for the Apple.

Anon, "Programming Tips," pg. 15-16.
A listing in Integer for this game on the Apple.

Micklus, Lance and Summers, Murray R., "Treasure Hunt," pg. 33-34.
Listing for this Adventure-type game.

Cross, Mark, "Bouncing Ball Catcher," pg. 46-47
An Applesoft program employing Hi-Res graphics.

Anon, "Switch Puzzle," pg. 50-51.
A game for the Apple.

Rubber Apple Newsletter (March, 1980)

Musgrave, J. E., "Change Disk Volume," pg. 2.
Short routine for the Apple machine language.

689. Recreational Computing 8, No. 5, Issue 44 (March/April 1980)

Wells, Arthur, Jr. "Recreational Apple II Hi-Res Graphics," pg. 4-8.
Lines, Triangles, and other shapes on the Apple.

Lindsay, Len, "Pet Games," pg. 11.
About 75 programs for the PET are reviewed.

Hatch, Larry, "Raging Robots," pg. 34-35.
Landmine the PET Screen to outsmart the robots.

Keyser, Earl, "Frogs for the Apple," pg 34-35.
Listing for the game "Frogs."

Gull, Steve, "Playing Simon on the PET," pg. 35
Try to duplicate the sequence of tones that the Apple plays.

690. Creative Computing 6, No. 3 (March, 1980)

Fricke, Victor, "Three Mile Island," pg. 38
Notes on running the popular nuclear power plant program.

Mecca, Lorraine, "The Computer Connection," pg. 58-59.
Contained in this article is a discussion of modems and the D.C. Hayes Micromodem II.

Cox, Ken, "PET as a Remote Terminal," pg. 60-62.
Notes on implementing a PET terminal program.

Howerton, Christopher, "Ches Clock," pg. 132-133.
Is speed chess your game? Use your Apple as a clock!

Carpenter, Chuck, "Apple-Cart," pg. 150 - 153.
Discussion of the use of POKEs, Applesoft READ...DATA, String Parsing, Text Typer, etc.

Yob, Gregory, "Personal Electronic Transactions," pg. 160 -163.
Discussion of PET Logic, Two's Complement Tutorial, Light Pen etc.

691. Personal Computing 4, No. 4 (April, 1980)

Neiburger, E. J., "Score Your Heart Attack Risk," pg. 48-50.
Run this program, then take no changes — get a checkup (Apple).

Wood, Don, "Word Processing with Your Apple," pg. 68 - 70.
Notes on Apple writer, Super-Text, EasyWriter, Personal Text Processor, Aptype, Text Editor Version 3.0, etc.

Nichols, John M., "Housebreaking Your New Pet," pg. 73-74
How to run programs written for the 8K PET on the newer 16K, 32K PET.

Kilobeud Microcomputing No. 40 (April, 1980)

Baker, Robert W. "Pet-Pourri," pg 9,14.
Notes on PET Basic.

Anon, "Ohio Scientific's Small Systems Journal,"
A continuation of a discussion of multiple user systems.

Hayek, Tom, "The Basic Programmer's Toolkit," pg. 34-35.
Have a look at this helpful utility for the PET.

Chamberlain, Bruce S., "Fast Apple Peripherals," Pg. 92-96.
How to interface high-speed serial printers to the Apple II.

Tenny, Ralph, "Get Started With MicroStart," pg. 118-128.
Hardware for experimenting with the 6502 and other CPU chips.

Tannenbaum, Larry M., "KIM Vari-Stepper," Pg 146.
An automatic variable speed single-stepper for the KIM.

David, D. J., "PET's Librarian," Pg. 172-173.
An automatic variable speed single-stepper for the KIM.

Gordon, Hal T., "Instruction Sets Examined and Compared," pg 174-180..
The second part of this article looks at on-chip and off chip registers.

Dunmire, Jerry, "Indexing for the PET," pg. 186-187.
A solution for the lack of a counter on the PET tape recorder.

Hitt, Peter G., "Build a Home for your Superboard II,"pg. 202-205.
House your OSI Superboard with a power supply.

693. **Byte 5, No. 4 (April 1980)**

O'Flaherty, John, "A White-Noise Generator for the Apple II," pg. 68.
A simple machine language program turns the Apple into a white-noise generator.

Chamberlin, Hal, "Advanced Real-Time Music Synthesis Techniques," pg. 70-94, 180-196.
Discussion of difficulties in computing waveforms fast enough for real-time music systems.

Cross, Mark A., "Apple Audio Processing," pg. 212-218.
The Apple is capable of playing several notes simultaneously with simple homebrew interfaces.

694. **Southeastern Software Newsletter, Issua 17 (April 1980)**

Staff, "Binary Search and Sort Modules for Names File," pg. 2-8.
Notes on searching and sorting on the Apple.

Chipchase, Frank D., "Better Utilization of Apple Computer's Renumber and Merge Program," pg. 8-9.
Set up a renumber and merge Binary file.

695. **OSI User's Independent Newsletter (January 1980)**

Curley, Charles, "Machine Language Memory Test," pg. 2.
How to adapt Jim Butterfield's Memory Test (First Book of KIM) to the OSI or other 6502 machines.

696. **MICRO, No. 23 (April 1980)**

Lacy, Allen J., "Applesoft II Shorthand," pg. 5-8.
This routine allows a programmer to type in an entire Applesoft command with the use of one control key.

Tripp, Robert M., "The Value of 16 Bits," pg. 9.
Notes on 8 bit vs. 16 bit processors.

Crouch, Bill, "The Apple Stripper," pg. 11-12.
Remove REM statements from your Basic program for the 'run version'.

Taylor, William L., "Graphics and the Challenger C1P, Part 4," pg. 16-19.
This installment shows how the previous material can be used to create pictures.

Blalock, John, M., "SYMple Basic Data Files," pgs. 21-25.
Implement Data Save and Data Load on your SYM-1.

Evans, Mel, "A Perpetual Calendar Printer for the AIM," pg. 27-29.
A few programming tricks are used in this AIM program.

Wagner, Roger, "Bi-directional Scrolling," pg. 23.
Scroll the Apple page down as well as up.

Hyde, Randall, "The SY6516 Pseudo-16 Bit Processor," pg. 36-37.
This new microprocessor extends the capabilities of the 6502 with some 16 bit operations and improved addressing, etc.

Strasma, Rev. James, "PET Keysort," pg. 43-56.
A complete general purpose keysorting program.

Swank, Joel, "KIM Scorekeeper," pg. 59-62.
A general purpose, multi-player scorekeeper that can be used as a part of larger game programs.

Morris, E.D., Jr., "OSI Basic in ROM," pg. 65-66.
To help you understand OSI Basic, a table of the locations of the subroutines to service the main commands is presented.

Rowe, Mike, (Staff) "The MICRO Software Catalogue: XIX," pg. 71-72.
Ten programs are described.

Dial, Wm. R., "6502 Bibliography: Part XIX," pg. 77-78.
Some 65 more articles are referenced.

697. **Rainbow, Issue 1 (January 1980)**

Rennard, Bob and Simpson, Rick, "Character Input and Output," pg. 4-5.
Information on Apple Machine Language.

Marcuse, D., "Applesoft ROM Card," pg. 4-5.
How to use the Apple mini-assembler with the ROM card installed.

Turnbull, Ernie, "Audio Monitor," pg. 5-6.
How to monitor your Audio from the Cassette while loading the Apple.

Butler, John W., "Geneology Program," pg. 6.
Keep track of 250-350 families on a single Apple diskette.

Simpson, Rick, "Mini-assembler and the Language System," pg. 7.
Use the language system and still have the services of the miniassembler, Apple.

Rennard, Robert, "Responses and Remarks," pg. 8-9.
Apple output registers, improved Apple documentation, etc.

Barnes, John, "An Applesoft Sound Routine," pg. 12.
Machine language entered by an Applesoft program for tone generation.

Mathes, Stan, "Make your Apple Sing," pg. 13.
Write a music program on the Apple.

Wagner, Roger, "The Reliable Applesoft Append," pg. 15.
Notes on using the Applesoft append program.

Anstis, Stuart, "Catch a Star," pg. 23.
Listing for an Apple Game.

698. **The Target (March/April)**

Anon., "Grapevine," pg. 1.
Changes to zero page usage in AIM-65 Basic.

Silber, Steve, "Short-Cut Auto-Number," pg. 2-5.
A modification of the Short-Cut program which provides for auto line numbering.

Bresson, Steve, "Slow Display," pg. 3.
Slow down the rate at which characters are displayed on the AIM display.

Sellers, George, "KIM-4 Motherboard," pg. 6.
Add a KIM-4 motherboard and RAM memory to your AIM-65 by a simple hardware modification.

Clark, Jim, "An AIM 65 Scanning Subroutine — SCAN," pg. 7.
Scan the keyboard to see if a key is pressed, get the key and respond to it in AIM programs.

Peterson, Gary, "Exterminating Some Invisible Bugs," pg. 8.
High voltage spikes can be removed at your AIM power supply by using a varistor and a few capacitors and diode devices.

Bresson, Steve, and Semancik, Bill, "Lunar Lander," pg. 9.
Real time lunar lander for the AIM.

899. Interface Age 5, (May 1980)

Rothman, Howard H., "Advertising with the Apple Computer," pg. 66-68.
Put your advertising message on the SCROLLING WONDER.

700. SoftSide 1, No. 3 (April 1980)

Ford, Robert, "Juggle," pg. 9-13.
Juggle a number of balls on this Apple graphics program.

Pachin, D., "Applesoft Ampersand," pg. 16.
Use the ampersand to initiate special routines.

Hill, Alan, "Shoot Out," pg. 19-22.
A two-person game for the Apple graphics.

Sander-Cederlof, Bob, "Jig-Saw Puzzle," pg. 28-29.
Try your hand at assembling a jig-saw puzzle on the Apple.

Crossman, Craig, "The Invisible Signature," pg. 32-33.
Put your own label within your Apple program and hide it!

Sander-Cederlof, Bob, "Space War," pg. 35-37.
A two-player graphics game for the Apple.

Kapur, Mitch, "Melody," pg. 40-47.
Create and save your melody using this Apple program.

Blackwood, George H., "Intimate Instructions in Integer Basic," pg. 49-52.
Starting a series of installments on detailed programming instruction for the Apple II.

Anon., "Display Control Characters," pg. 52.
Use this short routine to detect or reveal special characters in Apple programs.

701. Creative Computing 6, No. 4 (April 1980)

Lindsay, Len, "Atari in Perspective," pg. 22-30.
Comparisons of the pro's and con's of the PET and the Atari 800 micros.

Bradford, William, "Ten to the Thirty-Eighth," pg. 104-110.
Here is a game called GOOGOL for the Apple.

Carpenter, Chuck, "Apple-Cart," pg. 122-129.
A listing and discussion of a program, Simple File Builder.

702. Rainbow 2, Issue 2 (February 1980)

Headland, Rex, "Dollar Formatting Gosub," pg. 4-5.
A dollar formatting routine for the Apple.

Wagner, Roger, "Exceeding the Speed Limit with your Apple II," pg. 8.
How to speed up your Apple program.

Lipson, Neil D., "An Improved Hi-Res Light Graph," pg. 11-12.
Display up to five color graphs on the same screen.

Wagner, Roger, "Fast GR-Screen Clear," pg. 14.
A routine in either Integer Basic or Applesoft Basic.

Wagner, Roger, "An Unlikely Character," pg. 14-15.
Type strange characters with special key combinations on the Apple.

Wagner, Roger, "Append-Ectomies In Integer and Applesoft," pg. 15.
After joining binary data to the end of a program, use this routine to remove the appended part.

Deardon, Hinkley, W., "From the Pits," pg. 16-17.
Some notes on computer accessory advertising and supply practices.

Martin, Bill, "Crossing Your Wires," pg. 17-18.
Notes on running remote monitors, better audio, etc. Apple.

Busdiecker, Roy, "The Number Game: An Introduction to Computer Arithmetic," pg. 20-24.
All about those strange binary numbers together with a listing for a Decimal to Binary program.

703. Compute II, Issue 1 (April/May 1980)

Rehnke, Eric, "The Single-Board 6502," pg. 3-8.
Notes on an EPROM simulator, improved disk-based assemblers, Speak and Spell Interface, etc.

Zunchak, Gene, "Nuts and Volts," pg. 9-14.
All about the 6502 Read/Write timing, Interfacing, Access Time, etc.

Day, Michael E., "RS232 Communications: Part 1," pg. 16-18.
Learn all about the use of RS232 interfaces to connect communications devices together.

Stone, Harold, R., "An Upgrade for KIM Microchess 1.0," pg. 19-23.
Modify the Microchess 1.0 to play a better game of chess.

Ditts, Joseph A. and Herman, Harvey B., "Program Transfers (PET to KIM)," pg. 25-26.
Using this transfer routine you can use Basic PET programs on your KIM.

Isaacs, Larry, and Compute Staff, "Designing an IEEE-488 Receiver with the SYM," pg. 27-30.
Part 1: Implementing the IEEE-488 Bus on a SYM-1.

McCreary, Dann, "COSAPPLE, an 1802 Simulator for the Apple II," pg. 34.
COSAPPLE is an 1802 simulator and debugger designed to run on the Apple.

McCreary, Dann, "COSMAC: KIM-1 1802 Simulator," pg. 34.
This 1802 simulator is capable of real-time operations of moderate speed.

Sandlin, Larry, "Fun with the 1802," pg. 34-35.
Have fun with the low cost, low power consumption 1802.

Lock, Robert, "The Serious Side of the 1802," pg. 35.
Applications for the 1802 will be published in the future installments.

DeJong, Marvin L, "Improved Pulse Counting Software for the 6522 VIA," pg. 36-38.
Sharpen up your timing or frequency counter programs with the suggestions in this article."

Oliva, Richard F., "Printing a Symbol Table for the AIM-65." pg. 40.
In revising a program, a print-out of the symbol table can be very helpful."

Sproul, Keith, "Hard Copy Graphics for the KIM," pg. 43-46.
With a bit-mapped video board you can do professional quality graphics.

Mackay, A.M., "24 Hour Clock for SYM-1 Basic," pg. 46-48.
With this program you can have a time-of-day clock.

Stanford, Charles, L., "Screen Clear Routines for the OSI C1P," pg. 49-50.
Speed up the screen clear routines.

Missing MICRO Information?

MICRO is devoted exclusively to the 6502. In addition, it is aimed at useful, reference type material, not just "fun and games". Each month MICRO publishes application notes, hardware and software tutorials, a continuing bibliography, software catalog, and so forth. Since MICRO contains lots of reference material and many useful program, most readers want to get the entire collection of MICRO. Since MICRO grew very rapidly, it quickly became impractical to reprint back issues for new subscribers. In order to make the older material available, collections of the reprints have been published.

[A limited number of back issues are still available from number 7 to 18 and 20 to current. There are no 19's left.]

The BEST of MICRO Volume 1 contains all of the significant material from the first six issues of MICRO, from October/November 1977 through August/September 1978. This book form is 176 pages long, plus five removeable reference cards. The material is organized by microcomputer and almost every article is included. Only the ads and a few 'dated' articles have been omitted. [Now in third printing!]
Surface...$7.00 **Air Mail...$10.00**

The BEST of MICRO Volume 2 covers the second six issues, from October/November 1978 through May 1979. Organized by microcomputer, this volume is 224 pages long.
Surface...$9.00 **Air Mail...$13.00**

The BEST of MICRO Volume 3, covering the twelve issues from June 1979 through May 1980, will be over 400 pages long. It is scheduled for late summer 1980. The price is still to be determined.

For a free copy of the Index for Volumes 1, 2, and 3, please send a self-addressed, stamped envelope to:
BEST of MICRO, P.O. Box 6502, Chelmsford, MA 01824

Advertiser's Index

The Great American Solution Machine.

More than 50,000 students, teachers and administrators solve problems with this reliable Commodore computer.

You're looking at the Number One computer in education today.

In fact, you've probably already used it.

The Commodore.

You know it teaches. Guides. Challenges. Analyzes. Organizes. Simplifies.

But did you know it has capabilities that are far beyond its price range?

You can accomplish tasks with The Commodore at a price/performance ratio that leads the field.

You can also count on The Commodore showing up for class every day.

It's a remarkably sophisticated, remarkably reliable machine. Around the world in schools—and businesses too—there are more than 100,000 Commodore computers now at work.

If you sense a snag in the flow of knowledge in your classes, we think you should challenge The Commodore.

Compare it against any computer in—or above—its field.

See if it won't raise the level of interest and accomplishment among your students.

And simplify the complex in your administrative duties.

All at a price that makes it stand alone.

For the name of your nearest authorized Commodore dealer, just write to: Commodore Business Machines, Inc., 950 Rittenhouse Rd., Norristown, PA 19401

Call now toll-free. Ask for operator 973:

800-824-7888

(In Calif., 800-852-7777)
(In Alaska and Hawaii, 800-824-7919)

commodore

COMMODORE